DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943

# NAVAL POSTGRADUATE SCHOOL
## Monterey, California

# THESIS

SOFTWARE MAINTENANCE RELATING TO THE INPUT TRANSLATOR AND Z80 REALIZATION VOLUME OF THE COMPUTER SYSTEMS DESIGN ENVIRONMENT

by

Robert Ralph Vogel

March 1985

Co-Advisors: A. A. Ross
N. F. Schneidewind

Approved for public release; distribution is unlimited

T223076

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

## REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

| | |
|---|---|
| 1. REPORT NUMBER | 2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE *(and Subtitle)*: Software Maintenance Relating to the Input Translator and Z80 Realization Volume of the Computer Systems Design Environment | 5. TYPE OF REPORT & PERIOD COVERED: Master's Thesis March 1985 / 6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s): Robert Ralph Vogel | 8. CONTRACT OR GRANT NUMBER(s) |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School Monterey, CA 93943 | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS |
| 11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Postgraduate School Monterey, CA 93943 | 12. REPORT DATE: March 1985 / 13. NUMBER OF PAGES: 121 |
| 14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)* | 15. SECURITY CLASS. *(of this report)*: UNCLASSIFIED / 15a. DECLASSIFICATION/ DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution is unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Computer aided design, controller, CSDE, CSDL, Z-80 Primitive, realization, input translator

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This thesis corrects the discrepancies between the input Translator and the Z-80 Realization Volume of the Computer System Design Environment (CSDE). It also demonstrated for the first time, complete processing of a problem through CSDE. CSDE is a computer-aided design system for real time controllers. The Translator takes as input, a Computer System Design Language (CSDL) problem and generates a primititive list. Each primitive is matched to identically named primitive realizations (Continued

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601



SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

ABSTRACT (Continued)

in the Realization Volume. The final outputs are hardware and software listings to implement the initial design.

Approved for public release; distribution unlimited

# Software Maintenance Relating to the Input Translator and Z80 Realization Volume of the Computer Systems Design Environment

by

Robert Ralph Vogel
Lieutenant, United States Navy
B.S., United States Naval Academy, 1977

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
March 1985

## ABSTRACT

This thesis corrects the discrepancies between the input Translator and the Z-80 Realization Volume of the Computer System Design Environment (CSDE). It also demonstrated, for the first time, complete processing of a problem through CSDE. CSDE is a computer-aided design system for real time controllers. The Translator takes as input, a Computer System Design Language (CSDL) problem and generates a primitive list. Each primitive is matched to identically named primitive realizations in the Realization Volume. The final outputs are hardware and software listings to implement the initial design.

TABLE OF CONTENTS


I. INTRODUCTION ---------- 8

II. BACKGROUND ---------- 12

A. SYSTEM DESIGN ---------- 12

B. FOCUS OF THESIS ---------- 17

III. METHODOLOGY ---------- 20

A. DETERMINING MEANINGS OF NEW PRIMITIVES ------- 21

B. COMPUTER SYSTEM DESIGN LANGUAGE (CSDL) PHILOSOPHY ---------- 24

C. PRIMITIVE CONSTRUCTION ---------- 27

1. Input/Output ---------- 31

2. S.fixedwait ---------- 37

3. S.call ---------- 38

4. S.equivalenc and S.implicate ---------- 39

5. S.forcons ---------- 39

6. S.whilestart ---------- 41

7. S.in and S.ni ---------- 43

8. S.stboolwait and S.boolwait ---------- 43

9. S.waitleast ---------- 46

10. Changes to Smith's Primitives ---------- 48

IV. TESTING AND DEMONSTRATION ---------- 51

A. TESTING SEQUENCE AND EXAMPLE ---------- 52

B. TRANSLATOR ERRORS ---------- 55

C. THE CSDE PROGRAM ---------- 59

V. CONCLUSIONS ---------- 63

APPENDIX A - PRIMITIVES AVAILABLE FROM TRANSLATOR ----- 66

APPENDIX B - CSDL TEST PROGRAM ----------------------- 69

APPENDIX C - PRIMITIVE LIST, APPLICATION TIMING TABLE AND SYMBOL TABLE PRODUCED BY TRANSLATOR FROM CSDL TEST PROGRAM -------- 72

APPENDIX D - COMPARISON OF PRIMITIVES ----------------- 77

APPENDIX E - REVISED Z-80 REALIZATION VOLUME ---------- 81

APPENDIX F - WORKING CSDL TEST PROGRAM ---------------- 106

APPENDIX G - SUMMARY OF TRANSLATOR ERRORS ------------- 116

LIST OF REFERENCES ----------------------------------- 119

INITIAL DISTRIBUTION LIST ---------------------------- 121

LIST OF FIGURES

1. Current Ross Controller Design System----------12

2. Syntax Structure Corresponding to 's.fixedwait'------22

3. Primitive List Segment Containing 's.fixedwait'------23

4. CSDL Problem Segment Corresponding to Primitives in Figure 3----------24

5. CSDL Implementation of Input----------32

6. Changes Required for 's.whilestart'----------42

7. Changes Required for Boolean Wait Primitives---------45

8. Changes Required for 's.waitleast'----------47

9. Realization of Blocks of Code to be Executed Once Only----------49

10. Changes Required for Relational Primitives----------58

11. Contents of Global Variable File----------62

# I. INTRODUCTION

In the field of computer systems development, the current techniques are coming under increased scrutiny because of intolerably high costs. Hardware and software costs make up the two component parts of computer systems development costs. One source projects software to comprise approximately 90 percent of total system costs by 1985 [Ref. 1: p. 7]. This trend applies equally to large automated data processing systems and to special purpose microprocessor based systems, also called embedded systems. Total software costs for embedded computer systems, alone, in the Department of Defense(DOD) are projected to exceed 32 billion dollars by 1990 [Ref. 1: p. 8] with 40 to 75 percent of this total going to software maintenance. Since embedded computer system expenditures comprise roughly 30 percent [Ref. 2: p. 45] of all DOD software spending, this clearly illustrates that ways must be found to reduce these costs.

Current software/systems development methodologies generally embody a life-cycle approach with various phases such as requirements analysis,design, coding, implementation and testing. This process is expensive, time consuming and flawed mainly because the hardware choices are made early in the process to insure hardware availability upon

commencement of system testing. Consequently, misjudgments concerning hardware and software integration and the inability to completely satisfy the original requirements specification must either be 'lived with' at the end of the project or corrected at great expense.

Automated design tools of various types seem to hold the greatest promise in terms of increasing productivity of systems designers and programmers. They range in complexity from single function tools such as compilers, interpreters and editors to fourth generation languages, applications generators and complete software systems generators [Ref. 3: p. 63]. The key point of these latter types is that they greatly reduce the amount of labor required to finish a system design once the requirements specification has been completed. Although these tools may be primarily thought of as applying to large automated data processing projects, the principles also apply to embedded computer systems. Thus, similar tools exist and are being developed to aid designers of embedded systems, examples of which are real-time controllers and computers found in weapons systems guidance packages.

One approach to computer-aided design tools for software and systems design is rapid prototyping. Rather than going through the traditional phases of the design process and hoping that the single final product is on time, on budget, and satisfies the requirements specification, rapid

prototyping allows preliminary designs to be produced relatively cheaply and quickly. Changes to a first prototype can be easily incorporated into a second and the process continues until the desired results are achieved. The important point to be remembered is that the first step in any design, the requirements specification, will still require thorough research by the designer regardless of what design tools are employed. Rapid prototyping encourages the consideration of software and hardware simultaneously throughout the development cycle and should result in the optimum design of a microcomputer based product at the lowest cost [Ref. 3: p. 76].

A computer-aided design tool for rapid prototyping of microprocessor based real-time controllers has been in development at the Naval Postgraduate School since 1982. The Computer Systems Design Environment (CSDE) was originally implemented by Alan A. Ross [Ref. 4] in 1978 based on initial research by M.N. Matelan [Ref. 5]. CSDE has been the subject of several thesis efforts at NPGS, each examining a different module of the system. Currently, all components of the system have been completed, but certain conflicts between some of them required resolution before a successful demonstration of CSDE could be accomplished. The subject of this thesis was to identify and resolve the procedural conflicts that existed between certain

modules of the CSDE system and to complete a successful demonstration.

Concurrent to this research, two projects related to CSDE were also completed. Mr. Greg Lukas was hired to perform extensive software maintenance on the CSDE program to make it more user friendly and run more efficiently. LCDR Jim Poole worked on a project to streamline the physical process of running a design problem through CSDE.

Poole's research attacked a problem, documented by Smith [Ref. 6] and Riley [Ref. 9], in which numerous steps were required to work within the CSDE environment. The steps involved the separate uses of the VAX 11/780 minicomputer, ALTOS Z-80 microcomputer, the Pro-Log microcomputer, and data transfer via modem. Poole constructed a single Zenith Z-100 microcomputer workstation from which all CSDE operations could be conducted. This makes CSDE much more convenient for the user. [Ref. 12]

## II. BACKGROUND

### A. SYSTEM DESIGN

To better understand what the procedural conflicts were and where they specifically existed within CSDE, a review of the system is essential. Refer to Figure 1 [Ref. 6: p. 20] for a simplified block diagram.

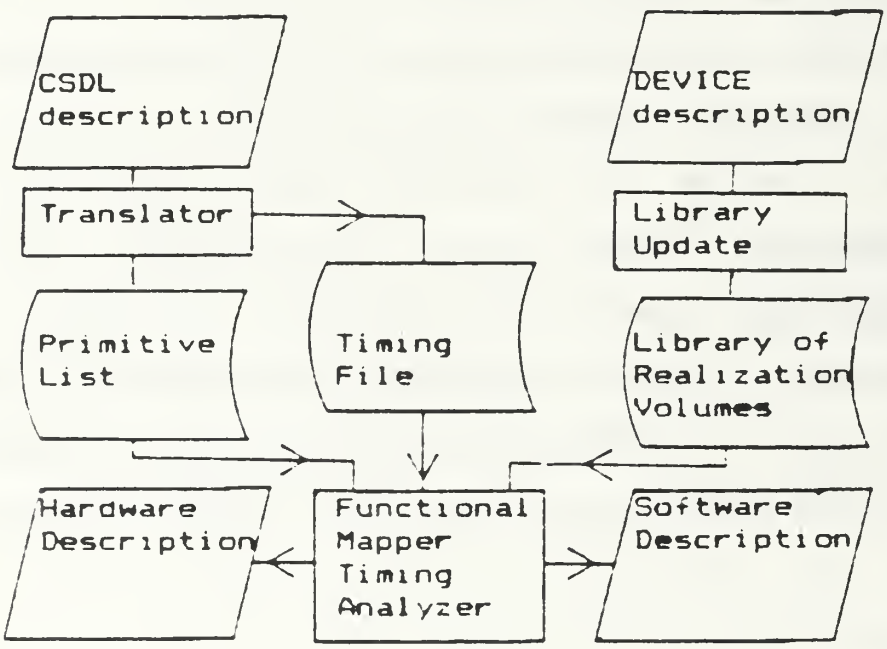


Figure 1 Current Ross Controller Design System

The basic concept of CSDE is that a designer be able to describe a controller application functionally in a high order language, input this description into the system and obtain software and hardware listings that describe a prototype implementation of the desired application.

Currently, three common microprocessors, the Zilog Z-80, Intel 8080 and Intel 8086, are available for hardware implementation depending on which microprocessor best satisfies the design requirements as determined by CSDE.

The following discussion relates to the various blocks in Figure 1. First, the problem statement or functional description of the controller is written in the Computer System Design Language (CSDL). The syntax used by CSDL was originally defined by Matelan and is summarized by Ross [Ref. 4: pp. 10-12]. A dedicated language manual for CSDL has not yet been developed. The CSDL problem statement includes such things as the variables to be used, functions to be executed and contingency/task pairs. A contingency/ task pair is simply a statement that describes which function or task will be executed in response to a particular condition or contingency. The problem statement is then input to the Translator which is equivalent to inputting a Pascal source program into a Pascal compiler.

The Translator is a compiler which reads the CSDL program and generates four output files. Two of these four files are required by the CSDE system and are shown in Figure 1. The Primitive List is output in a file named 'PRIMFILE.DAT' and contains a list of primitives which describe the input problem. A primitive is basically a macro-instruction which is later translated by CSDE into assembly language instructions for the chosen

microprocessor. An example of a primitive is 's.jmpf' which causes a jump to a location if a variable is false.

The Timing File, also called the Application Timing Table, is output in a file named 'IADEFL.DAT'and contains attributes of the contingency/task pairs such as maximum allowed time duration of each task and contingency and the relative priority of each pair. This information is stored in a table format and is used by the Timing Analyzer to produce a monitor program for the selected microprocessor. The monitor program (similar to a simplified operating system) insures that all contingency/task pairs are executed within the required time constraints as stated in the original CSDL problem.

The third and fourth output files from the Translator are named 'SYMFILE.DAT' and 'TRANSLATE.DAT'. 'SYMFILE.DAT' contains the Symbol Table, also called the Environment Table. The Symbol Table is a listing of attributes of variables and constants such as type, precision, and value. The Symbol Table is actually a subset of the Primitive List ('PRIMFILE.DAT'). It's use by the CSDE program is optional. If SYMFILE.DAT is available, the Functional Mapper will read the Symbol Table before reading the entire Primitive List and the CSDE program will execute faster. 'TRANSLATE.DAT' is a text file for user convenience and is not used by CSDE. Carson used it as an aid in debugging during the development of the Translator. Currently, diagnostics which trace

errors in Translator input, will deposit error messages in this file [Ref. 8: p. 28].

The key aspect of CSDE is how it picks the particular microprocessor to be used and how it generates the assembly code listing to make the system work according to the requirements of the initial problem statement. The primitives generated by the Translator are really generic in nature. To convert them to assembly code, each primitive is matched to an identically named primitive in the Realization Volume that applies to the specific microprocessor that has been selected to implement the problem. For example, the Realization Volume for the Zilog Z-80 contains a list of primitives and the Z-80 assembly code sequence that implements each primitive. Thus, the Functional Mapper would take the primitive, 's.jmpf', which had been produced by the Translator and match it to 's.jmpf' in the Z-80 Realization Volume. The primitive, 's.jmpf', is implemented or realized with three Z-80 assembly language statements. This matching process is called functional mapping because the Translator primitive is mapped into the Z-80 Realization Volume. Details relating to variables, precision, and timing are addressed by Ross [Ref. 4: pp.79-85].

The Library of Realization Volumes (Figure 1) currently contains three volumes based on three microprocessors. An Intel 8080 Volume was developed by Ross during his original research and an Intel 8086 volume was added by A. J. Cetel

[Ref. 7]. The Z-80 volume, added by Smith [Ref. 6], was chosen for use during this thesis research because the prototype computer used for demonstration is currently configured for Z-80 operation.

Device Description and Library Update (Figure 1) refer to the process of adding new Realization Volumes in the future as well as updating currently existing volumes. This process is extremely complex because a Realization Volume not only includes software primitives and their assembly language translations, but also contains hardware primitives which describe the chips or multi-chip circuit boards required to implement the software primitives. In the Z-80 Realization Volume, all hardware primitives refer to circuit boards rather than individual chips because Smith designed his Volume for the Pro-Log computer. The Pro-Log can be reconfigured easily by installing different boards as required [Ref. 13]. An example of a Z-80 hardware primitive is 'h.atod' which calls for an 8 bit analog to digital conversion board. The actual translation of 'h.atod' specifies items like which circuit board to use and which jumper pins should be connected or disconnected.

The hardware listing produced by CSDE for a given design is the indirect result of the software primitives generated by the Translator. As Translator-produced software primitives are successfully mapped to software primitives in a Realization Volume, hardware is also specified.

Translations of software primitives in the Realization Volume contain references to hardware primitives, also contained in the same Realization Volume. For example, the translation for a software primitive like 's.clockon25' (generate a 25 millisecond clock) contains the statement, 'include h.clock'. 'h.clock' is the Z-80 hardware primitive which details the connections for the counter-timer chip on the CPU circuit board to produce a 25 millisecond clock. Modifications or additions to software primitives in a Realization Volume must also be accompanied by appropriate changes to hardware primitives. This insures that a realizable controller design, in terms of software and hardware, can still be produced by CSDE.

## B. FOCUS OF THESIS

The Translator module of the CSDE system was not developed by Ross during his initial research. During subsequent thesis research at NPGS the Translator module was written by T. H. Carson [Ref. 8]. Concurrent to Carson's work, the Z-80 Realization Volume was developed by Smith [Ref. 6]. A successful demonstration of the complete CSDE, from input to operation, is the subject of this thesis. Previous discrepancies between these two modules prevented such a demonstration.

The discrepancies involved differences between the primitives that were produced by the Translator and

primitives that were available for use in the Z-80 Realization Volume. For example, the Translator generates a primitive called 's.sensecond' which relates to sensing a condition or testing for a certain flag. There was no 's.sensecond' primitive in the Z-80 Realization Volume. This did not mean that the Z-80 Volume lacked primitives that could implement the function of 's.sensecond'. Rather, the problem may only have existed in the names used to label the same macro-instruction. This problem of discrepancies between primitives generated by the Translator and those available in the Z-80 Volume caused CSDE to abort an attempted controller implementation with a Z-80 microprocessor when such a discrepancy was encountered.

To correct the discrepancies with primitives, all primitives available from the Translator were compared to those present in the Z-80 Realization Volume. For Translator primitives like 's.sensecond', which had no matching Z-80 primitives, a solid understanding of the function of the primitive was gained. Then the Z-80 Volume was examined to see if the function in question was labeled with a different name or implemented in a different manner. Corrective actions consisted of modifying old primitives and adding new primitives to the Z-80 Realization Volume so that all primitives produced by the Translator can now be realized except for 's.in/outport'.

Once the discrepancies with primitives were corrected, sample demonstration problems were run through the entire process to test each new primitive and those old Smith primitives retained in the revised Z-80 Realization Volume. The demonstration problems were simple and only proved that each primitive could be processed by the CSDE system based on a single set of data. This brings to light the general problem of systems testing within the computer industry. Just because one problem was successfully demonstrated in CSDE does not mean other errors do not exist. However, exhaustive testing was not possible during the scope of this research just as it is rarely possible in industry.

## III. METHODOLOGY

The task of identification and correction of discrepancies between output from Carson's Translator and primitives available in Smith's Z-80 Realization Volume is a difficult software maintenance project. The importance of well documented code and the desirability of a face to face turnover between past and current researchers was made painfully clear as work progressed. The primary sources of information regarding CSDE were more than adequate with the availability of Matelan's reports, Ross's thesis, and LtCol Ross, himself. Information regarding Carson's Translator consisted of his thesis, a loose sheet summarizing Translator produced primitives (Appendix A), and a CSDL test program (Appendix B) designed to produce a primitive list (Appendix C) containing all primitives available from the Translator. These last two items proved invaluable in helping to determine the functional meanings of the Translator produced primitives. Information regarding Smith's Z-80 Realization Volume consisted of his well documented thesis and a simple but important demonstration problem written and discussed by Riley [Ref. 9: Appendix E.].

Initially, a comparison was made between software primitives in the Z-80 Realization Volume and those

available from the Translator for one to one correspondence. The net result was that 15 of 39 possible primitives available from the translator had no matching primitive in the Z-80 Realization Volume. Thus, the first task was to write new primitives for the Z-80 Realization Volume to match the outstanding 15 from the Translator.

Smith wrote 68 software primitives for his Z-80 Realization Volume. It would seem to be a fairly reasonable task to write 15 more. Some were relatively easy while a few were not constructed for reasons discussed later.

## A. DETERMINING MEANINGS OF NEW PRIMITIVES

The general approach to writing a new Z-80 primitive is to first examine the part of the CSDL language that the primitive represents and to understand what it means. Since no language manual exits for CSDL, one must examine the actual syntactic structure that corresponds to the primitive. Carson made this correspondence through the use of production numbers. Each CSDL primitive listed in Appendix A has a production number which corresponds to the production number of its syntax structure. The CSDL syntax structures are displayed in Backus-Naur form which is a standard representation of the syntax structures of a computer language [Ref. 10: p. 16]. Originally conceived by Matelan, CSDL was refined by Carson when he developed his Translator and is displayed by production number in his

thesis [Ref. 8: pp. 47-54]. The syntax structure for a given primitive serves as the basis for development of a new primitive realization. Newly developed primitives are discussed individually in subsection C of this chapter and the CSDL syntax structures that apply are listed as each one is discussed.

In most cases, the Backus-Naur syntax structures were insufficient to determine the meanings of new primitives. One also had to study the applicable portion of Carson's CSDL test program and look at the corresponding set of primitives. In this manner one could see the context in which the primitive was used to better determine its meaning. For example, to determine the meaning of 's.fixedwait' first look at its syntax structure in figure 2. Note that a word not enclosed in brackets is called a

```
<WAIT> ::= WAIT <PERIOD>
         / WAIT <EXPRESSION> : <PERIOD>
<PERIOD> ::= *NUMBER* <TIME MEASURE>
<TIME MEASURE>  ::= H / M / S / MS / US / NS
```

Figure 2 Syntax Structure Corresponding to 's.fixedwait'

terminal and a word that is enclosed in brackets, < >, is called a nonterminal. Terminals appear in a CSDL problem as is, while nonterminals are located elsewhere in the Backus-Naur description of CSDL until they ultimately are defined

as terminals. In this case <PERIOD> can ultimately be a number and measure of time ranging from hours to nanoseconds. The syntax for 's.fixedwait' is relatively straight forward but the format for construction of a primitive has still not been made clear.

The next step is to look at the primitive list generated by the Translator from the CSDL test program to find 's.fixedwait'. A segment of the primitive list containing 's.fixedwait' is shown in Figure 3. This primitive list was

```
      P  33t.generated for: KBINPMAIN     ********
      P  34s.proc       (KBINPMAIN:)
      P  35s.assign     (MENU,@C02:8,8)
      P  36s.issuevent  (MENU:8)
      P  37s.sensecond  (KEYCHAR:8)
      P  38s.eq         (@T01,KEYCHAR,@C01:8,8,8)
      P  39s.jmpf       (@T01,@02:8)
      P  40s.assign     (MINTAC,@C01:8,8)
      P  41s.loc        (@02:)
---> P  42s.fixedwait  (10)                        <---
      P  43s.exitproc   (KBINPMAIN:)      ********
```

Figure 3 Primitive List Segment Containing 's.fixedwait'

generated for TASK KBINPMAIN in the procedures section of the CSDL test program, shown in Figure 4. By looking back and forth between the these two figures one can understand each CSDL construct and its matching primitive. In the case of 's.fixedwait', 'WAIT 10MS' results in the primitive 's.fixedwait (10)' and means that when the 'WAIT' instruction is encountered no other tasks are to be executed or contingencies checked until the specified time has

elapsed. This definition of 's.fixedwait' was confirmed by LtCol Ross prior to construction of the Z-80 primitive. He

```
      TASK KBINPMAIN;
              MENU:=0; ISSUE (MENU);
              SENSE (KEYCHAR);
              IF KEYCHAR=1 THEN MINTAC :=1; END IF;
--->          WAIT 10MS;                                <---
      END KBINPMAIN;
```

Figure 4

CSDL Problem Segment Corresponding to Primitives in Fig 3

was consulted prior to construction of all other primitives in question because of his familiarity with Matelan's concepts of the CSDL language.

A final source of information to be checked prior to actual Z-80 primitive construction is one or both of the other two currently existing realization volumes. For example, Ross wrote an 's.fixedwait' primitive for his Intel 8080 Realization Volume [Ref. 4: p. B-4]. This provided an excellent model from which to work with good examples of the proper placement of CSDE statements as well as the assembly language statements that would cause the controller to execute the function of this primitive.

## B. COMPUTER SYSTEM DESIGN LANGUAGE (CSDL) PHILOSOPHY

Before discussing actual primitive construction an important question needs to be answered. Why only construct primitives in the Z-80 Realization Volume to match existing primitives available from the Translator instead of changing

the Translator to produce primitives to match the ones originally available in the Z-80 Realization Volume?

The philosophy of CSDL is that the designer should be able to specify the functional design of a controller totally independent from the knowledge of any specific hardware that might be used to build it. The structures available in CSDL are generic in nature and allow the description of arithmetic, logical, and input/output operations that could be applied to any computer-based controller. Thus, the primitives that are available from the Translator are a direct reflection of the CSDL language as originally defined by Matelan. To change the Translator so it would produce a primitive such as 's.atod' would first require a change to CSDL. A syntax structure would have to be added such that a designer could specify an analog to digital signal conversion during an input/output operation. Why did Smith include an 's.atod' primitive in the Z-80 Realization Volume?

Smith included many primitives whose functions are not supported by CSDL. Some, like 's.atod', were written to provide the capability to use the hardware that was available to him. In this case, one of the boards currently available is a Pro-Log compatible analog to digital conversion board, a Mostek mdx-a/d8 board. Another is 's.clockcons', written to enable use of the 3-channel clock that is co-mounted with the cpu on the Z-80A cpu board. As

it stands now, an analog to digital function can not be included in a controller designed within the scope of CSDE and would have to be added externally once the CSDE controller design was realized.

In addition to hardware specific primitives, Smith also included specialty primitives such as 's.consfp', 's.varfp', and 's.fptoieee' to handle internal data represented in floating point notation. Here too, CSDL does not allow the designer to be this specific.

The net result is that the modified Z-80 Realization Volume produced by this thesis research excludes many of Smith's primitives. If the CSDL language is expanded in the future some of these deleted primitives might then be included.

The final reason for not making any changes to the Translator is the lack of expertise of this researcher in the area of compiler design and construction. This also makes it impossible to correct a few formatting errors that occur in the Translator's output files. These formatting errors cause the CSDE program not to accept a logically correct CSDL problem straight from the Translator. The specific errors discovered are discussed in Chapter 4 and summarized in Appendix G.

## C. PRIMITIVE CONSTRUCTION

Having determined the meaning of the new primitive to be constructed, the first step in construction was to write the Z-80 assembly language routine that would accomplish the desired function. The routine was then tested independently to eliminate logical and syntax errors prior to inclusion in the rest of the primitive body. Although the routine could be tested separately on any CP/M Z-80 microcomputer, it was easily tested right on the Pro-Log utilizing the Zenith Z-100 workstation set up specifically for CSDE research. This workstation consists of the Z-100 connected to the Pro-log and connected to the VAX 11/780 on which the Translator and the CSDE program both reside [Ref. 12].

To test a routine, it was first written on the Z-100 using an editor or word processor like Wordstar in non-document mode. It was then assembled and linked using the procedures set forth by LCDR Jim Poole, developer of the CSDE work station [Ref. 12]. The starting address specified at time of linkage must be 4000h since this is where user addressable RAM starts in the Pro-Log. Once the routine was linked its resultant hex file was downloaded to the Pro-Log using the AMDS program resident on the Z-100 and on an EPROM starting at 0000h in the Pro-Log. Detailed procedures are contained in Poole's thesis. Having downloaded the routine into the Pro-Log, it was run by pressing the reset button on the Pro-Log then typing G4000 on the ADM-3 monitor which is

also connected to the Pro-Log. Results were checked by inspecting the contents of appropriate memory locations, again using the facilities of the AMDS monitor in the Pro-Log. This same procedure was used to test programs generated by CSDE except that instead of creating the assembly routine on the Z-100, the assembly program output from CSDE was downloaded from the VAX to the Z-100.

In the case of 's.fixedwait', for example, the assembly code consists of two down counter loops, one nested inside the other, that delays the cpu from doing anything else as its executing the loops. For demonstration purposes, it was linked to a short routine to display some letters on the Pro-Log once the software delay had been completed. The delay was written to handle time in milliseconds vice microseconds so that the user could see the delay by timing the interval with a watch from the time the routine was started until the letters actually appeared on the Pro-Log display. The reference for Z-80 assembly language was Zaks' Programming the Z-80 [Ref. 14] and the reference for writing the letter display routine was the users manual for the Pro-Log 7303 Keyboard/Display Card [Ref. 15: p.3-5].

Once the assembly language routine was written and tested the rest of the primitive was constructed. There are very strict formatting requirements for the construction of a primitive. Detailed instructions are contained in Ross's thesis [Ref. 4: pp. 79-85.] and particular attention must be

paid to the column dependencies and format for the arguments contained in the primitive title line. The best way to understand primitive construction format is to study previously written primitives by Ross [Ref. 4: Appendix B] and Smith [Ref. 6: Appendix C].

When the primitive was finished it was added to the Z-80 Realization Volume. The Volume is normally contained in a file called RELIZE.MAC although the CSDE program allows the designer to name the required files anyway he chooses. Its format is strict in that the first portion of the Volume must be an index of the primitive title lines in alphabetical order. The last 4 numbers following the second colon of each title line specify the first occurrence of a CALC line, the first occurrence of an ATTR line and the beginning and ending line numbers locating the primitive within the Volume. To make the addition or deletion of primitives to the Volume easier, a program is available on the VAX called FORMAT.EXE. It takes as input just the primitives, stripped of line numbers and with no index. The primitives must start in column 6 and the last 4 numbers specified in the primitive title lines can be delineated simply as spaces or asterisks such as in 's.fixedwait (time:0,1275:15,-5,18, , ,****,****)'. If certain values in a title line are to be left intentionally blank such as in 'h.cardcage (:: , , ,0,0,****,****)', the blanks must be present before input to FORMAT or else FORMAT will inject an

extra comma into the primitive title line. The FORMAT program will create the index, add line numbers, and correctly fill in the last 4 numbers in each primitive title line. To use the formatter, type RUN FORMAT. It will ask for an input file name which is the file containing the unnumbered primitives. Then it will ask for an output file name which should usually be RELIZE.MAC. The resultant output file is ready for use by the CSDE program.

Because of the variation in complexity of the new primitives, it is important to discuss specific aspects that make each one unique. This should help future researchers if CSDL or the Translator is modified and more primitives need to be added to the Realization Volume. The following primitives were added to the Z-80 Realization Volume: s.inputport, s.sensecond, s.outputport, s.issuevent, s.fixedwait, s.call, s.equivalenc, s.implicate, s.forcons, s.whilestart, s.in, s.ni, s.stboolwait, s.boolwait, and s.waitleast. These new primitives along with relevant primitives from Smith's original Z-80 Realization Volume can be found in the revised Z-80 Realization Volume in Appendix E. Each of the constructed primitives is discussed below. The primitive, 's.in/outport', was not constructed. The Z-80 Realization Volume now contains a realization of every primitive that can be invoked by the Translator except for 's.in/outport'.

1. Input/Output

Five different primitives relating to input/output (I/O) are produced by the Translator which were not present in Smith's Z-80 Realization Volume. They are 's.inputport', 's.sensecond', 's.outputport', 's.issuevent', and 's.in/outport'. A discussion of Matelan's philosophy towards I/O is appropriate before describing primitive details.

Matelan stated that 'type' refers to how data may be used rather than how it is stored. Traditionally, we think of types in the latter sense, such as a variable being specified as type integer or real. CSDL allows 3 transmission types: INPUT ONLY, OUTPUT ONLY, and DUPLEX (input and output) [Ref. 10: p. 18]. To understand the differences a real world example is presented.

If a controller were to periodically sense the position of a valve, the valve would need to produce a proportional analog output signal which would then be converted to a digital signal by an analog to digital converter. This digital signal would be available via a specific line to a specific port which would be uniquely addressed by the cpu of the controller. This port would be for INPUT ONLY with data on the current valve position always available.

A typical hardware I/O implementation for a Z-80 microprocessor might involve using a parallel input/output

interface chip (PIO), which provides 2 or 3 ports. A port, here, refers to an 8-bit connection that may be used for input or output [Ref. 11: p. 162]. A PIO chip is programmable to set the direction for which the ports will be used. To provide INPUT ONLY, the software must first select the port to which the position signal is sent and program the PIO so that the selected port is set up in the INPUT direction. Once the port is set up, information can be read from it as often as the programmer desires with a simple instruction such as the Z-80 "in a,(n)", where 'n' is the port address and 'a' is the accumulator.

The primitive, 's.inputport', comes from the CSDL syntax, <INPUT SPEC>::= INPUT:<TRANSMISSION BODY> END INPUT, and the primitive, 's.sensecond', comes from the CSDL syntax, <DATA INPUT>::= SENSE (<NAME>). Using the valve example, source to and primitives generated by the Translator might appear as in Figure 5. The primitive,

| SOURCE<br>(CSDL Problem Statement) | OBJECT<br>(Primitive List) |
|---|---|
| . | . |
| . | . |
| ENVIRONMENT | . |
| INPUT: VLVPOS,8,TTL | s.inputport (VLVPOS,TTL:8) |
| . | . |
| . | . |
| PROCEDURES | . |
| FUNCTION VALVCHEK | . |
| SENSE (VLVPOS) | s.sensecond (VLVPOS:8) |
| . | . |
| . | . |

Figure 5 CSDL Implementation of Input

's.inputport', would be used to set up the desired port for input (i.e. send appropriate control code to the PIO) and 's.sensecond', would be used to actually get data from the desired port into the cpu.

Similarly, the controller should be able to send a digital output signal via a digital to analog converter to the valve positioning motor. This would be an OUTPUT ONLY function. The primitive, 's.outputport' comes from the CSDL syntax, <OUTPUT SPEC>::= OUTPUT:<TRANSMISSION BODY> END OUTPUT, and 's.issuevent' comes from <DATA OUTPUT>::= ISSUE (<NAME>). The primitive, 's.outputport', would be used to set up the desired port for output (i.e. send the appropriate control code to the PIO) and 's.issuevent' would be used to actually send the digital valve positioning signal to the desired port for output.

The implementation of data transmission type DUPLEX is more complex. Duplex means the capability of INPUT AND OUTPUT through a single port with no prior set up. Theoretically, this might imply using a PIO that did not require control codes to set up the direction of one of its ports prior to using that port. Data could be 'sensed' from or 'issued' to a port defined as duplex without having to worry if the port was configured correctly. In the event that duplex was not implemented solely through the use of a non-programmable PIO, it might be achieved by including code within the 's.issuevent' and 's.sensecond' primitives

to change port direction whenever an output or input was required. Under what conditions would a designer desire a DUPLEX transmission type? Perhaps there are hardware restrictions that limit the number of ports available to the controller and DUPLEX is the only way to satisfy all of the I/O requirements of the external device being controlled. The CSDL philosophy is that the designer is not concerned with hardware when he originates his design. Hardware is determined by the realization volume, whose software primitives are written for a fixed set of components specified by the hardware primitives. If a designer inputs one design into CSDE with too many INPUT ONLY or OUTPUT ONLY data transmission types, CSDE might declare the design not realizable for a given microprocessor type such as contained in the Z-80 Realization Volume. In the case of the Z-80 Volume, hardware is specifically configured at the board level and I/O software primitives must incorporate the programming guidelines of the board manufacturers. Alternately, if the designer resubmitted his design using more DUPLEX types in lieu of separate INPUT and OUTPUT types, the design might be realizable.

There are no PIO devices installed in the hardware currently allowed by the Z-80 Realization Volume. If a PIO chip was installed, a DUPLEX data transmission type primitive could be added to the Z-80 Volume by constructing the 's.in/outport' primitive such that it emulates the

DUPLEX function. One possible scheme might be that when a DUPLEX type is declared, it is automatically set up to a default mode of INPUT only. Data may be 'sensed' from the referenced port with no change in port setup. If data is 'issued' to this port than a software mechanism must exist to reset the port for OUTPUT first and, upon completion of the data output, reset the port to its default configuration of INPUT ONLY.

In the case of the Z-80 Realizaion Volume, a single data port on the Pro-Log keyboard/display card is available for input or output. A single control port is also available to control the mode of display of data once it has been output from the CPU to the keyboard/display card. Data sent to the control port controls the output display and not the data port. Thus, this single data port is DUPLEX in nature since no control codes are required to configure it for input or output. However, no DUPLEX function, i.e. an s.in/outport primitive, has been added to the Z-80 Volume. Because of the simple hardware available, it would merely duplicate the functions of the 's.inputport' and 's.outputport' primitives. More importantly, a flaw exists in the Translator which will not allow data to be 'sensed' or 'issued' through a variable that has been declared as DUPLEX.

To demonstrate the concepts of I/O for this thesis, the I/O primitives have been constructed very simply. The

hardware they use is the Pro-Log keyboard/display board which only has one data port addressed at d0h. The value of the digital signal available at the port can be determined by inspection of the 8 leds on the board, representing bits 0 through 7 read right to left. The 's.outputport' primitive creates an output variable used to hold the output value, clears the led display, and sends a control code to the board's single control port, d1h, to disable the alphanumeric display. The 's.issuevent' primitive outputs the contents of the output variable declared by 's.outputport' to the data port. The value can then be determined by inspection of the 8 leds.

The primitive, 's.inputport', creates an input storage location. No other actions are required because of the very limited I/O facilities of the Pro-Log keyboard/display card. With only one data port available for input, no control codes are required. When more complex I/O hardware is available this primitive will require modification. The primitive, 's.sensecond', is slightly artificial in that a conversion routine was added to accommodate the 2 rocker switches on the keyboard/display card. These switches directly control the contents of bits 6 and 7 (assumes bits numbered 0 to 7) of the 8-bit data port when it is 'sensed' for input. Thus by masking out all but the left most 2 bits, 4 possible values can be 'sensed' for input as directly controlled by the position of the rocker

switches. Four possible input values were sufficient for the demonstration problems attempted. Exact details as to switch positions can be found in the comment lines of the primitive in Appendix E.

2. S.fixedwait

The primitive, 's.fixedwait', has already been discussed in section III.A. There are a few additional points of interest.

This primitive contains an attribute line in addition to comment and calc lines. Attribute lines are used when the length of time of execution of a primitive is directly related to one of its input values. In this case, the input value is the desired time delay in milliseconds and also controls the length of execution of the primitive. Normally, the maximum execution time for a given primitive appears in the primitive title line as the second number after the second colon. The units are in clock cycles. A negative number is treated as a flag to indicate that the actual attribute needs to be calculated at code generation time. The flag is also the offset value, starting from the primitive title line, of the line number where the attribute calculation can be found.

In the case of 's.fixedwait', the attribute line is 'attr time =<time>*4000' and means take the input value and multiply it by 4000, with the result being the real execution time for this primitive in terms of clock cycles.

The factor of 4000 comes from the fact that 1 millisecond equals 4000 clock cycles with a 4 Mhz clock. The Z-80A supplied on the Pro-Log cpu board uses a 4Mhz clock.

There is a format error in the way the Translator produces this primitive. Specifically, there is a missing colon; the format should be 's.fixedwait (10:)' instead of 's.fixedwait (10)'. Normally the missing colon would cause a fatal error within the CSDE program but the program has been modified to accept 's.fixedwait' without the colon. Informative error messages still result to remind the designer that the Translator requires modification.

3. S.call

The primitive, 's.call', comes from the syntax, <PERFORM TASK> ::= *ID*, and gives the designer the ability to execute one task from inside another task without first having to check the other task's associated contingency. The Z-80 code to implement this primitive consists of a call instruction to a label that marks the desired task's subroutine. Although logically correct, this primitive is not useful within CSDE because no mechanism exits to account for the extra run time that is incurred when a task is executed in this manner. If 's.call' is used, CSDE will not have an accurate execution time statistic for a given design and could falsely generate a realization of a problem that does not meet the designer's timing requirements.

4. S.equivalenc and S.implicate

Both of these primitives perform logical comparisons between two expressions in the same manner as 's.or' or 's.and'. The inputs to them are the boolean results of 2 expressions, true (ff hex) or false (00 hex). The output is a boolean result according to the specified truth table [Ref. 16: p. 81]. Truth tables for both primitives can be found in Appendix E in the comment lines for each primitive. The CSDL syntax for 's.equivalenc' is '<EXPRESSION> ::= <EXPRESSION> == <EXP_2>' and for 's.implicate' is '<EXP_2> ::= <EXP_2> => <EXP_3>'.

5. S.forcons

This primitive marks the top of a for-loop and was already included in Smith's Realization Volume under the primitive name 's.forstart'. Statements have been added to properly account for changes in execution time that arise depending on how many times a for-loop is executed. Also, the arrangement of arguments within the title line is slightly different from the format output from the Translator because of the rules by which primitive realizations must be constructed. The CSDL syntax is '<FOR HEAD> ::= FOR *ID* FROM <EXPRESSION> TO <EXPRESSION> : <MAX LOOP COUNT>'.

To accurately keep track of total execution time during loop operations Ross incorporated the global variable, 'reps', in the CSDE program. The value of 'reps'

is the factor by which the execution time of each primitive is multiplied prior to being accumulated by the timing analyzer. It is normally set equal to 1 but in the case of a loop it is set equal to the value of max loop count specified by the designer. Once the 's.forend' primitive is encountered, the value of 'reps' is reset to the value it had upon entering the for-loop. Previous values of 'reps' are saved and recovered through the use of a stack inside the CSDE program. This stack is only used for the global variable 'reps' and manipulated with the statements, 'calc push reps' or 'calc pop reps'. This stack arrangement allows the construction of nested loops.

The format of the realization title line is slightly different for the primitive format generated by the Translator because of an error related to the positioning of the value for max loop count. Currently the Translator output appears as follows:

s.forcons (COUNT,@C02,@C04,@03,@04:8,8,8,120).

The criteria section of the title line (to the right of the colon) should contain only values corresponding to the variables in the argument section (to the left of the colon). In this case there are 3 variables and 2 labels in the argument section and as such there should only be 3 values in the criteria section. The value, 120, is the specified max loop count and should be in the argument section instead of the criteria section in order to be

available within the primitive. Consequently, the primitive realization was constructed to accept the following format:

s.forcons (COUNT,@C02,@C04,@03,@04,120:8,8,8).

The max loop count value also appears in the 's.forend' primitive as generated by the Translator. It is not required for proper construction of a for-loop and is ignored. Manual editing of the primitive list output file from the Translator is required whenever the for-loop primitives appear. Otherwise a fatal error will result upon running the CSDE program.

6. S.whilestart

This primitive is used in conjunction with the primitives, 's.whilecon' and 's.whend' to construct a while-do-loop. Its functions are to establish a label for the beginning of a while-do-loop and manipulate the max loop count using the global variable, 'reps'. This is similar to the method used in 's.forcons'. The CSDL syntax is '<WHILE> ::= WHILE'. Again, there is a problem as to where the max loop count is placed.

The CSDL syntax currently calls for max loop count to appear in the 's.whilecon' primitive and not in 's.whilestart'. This is wrong because the condition to be checked to determine if the while-do-loop should be continued appears as other primitives between 's.whilestart' and 's.whilecon'. Thus, these primitives, as well as the primitives appearing between 's.whilecon' and 's.whend', are

executed for every repetition of the loop. For correct timing statistics the execution times for both the condition checking primitives before 's.whilecon' and the action primitives after 's.whilecon' be must multiplied by max loop count. This requires that the max loop count value appear in 's.whilestart'. The primitive realizations have been written accordingly. Editing of the primitive list output from the Translator is required whenever the while-do-loop primitives are generated. Examples of the changes required are shown in Figure 6.

Sample Output From Translator (unedited)

```
P  31s.whilestart(@03:)
     .
   (condition checking primitives)
     .
P  33s.whilecon  (@T01,@04:4)     <--- '4' is max loop count.
     .                                  Precision of @T01 is
   (action primitives)                  missing.
     .
P  38s.whend     (@03,@04:)
```

Required Format for Input to CSDE Program

```
P   31s.whilestart(@03,4:)          <---Max loop count here.
     .
    (condition checking primitives)
     .
P   33s.whilecon  (@T01,@04:8)      <---Precision of @T01
     .                                  added.
    (action primitives)
     .
P   38s.whend     (@03,@04:)
```

Figure 6 Changes Required For 's.whilestart'

7. S.in and S.ni

These primitives are produced when a timed block is specified. A timed block is a nested set of actions within a task or function, with its own timing criteria. This timing criteria is in addition to any criteria specified for the parent task or function in the application timing table. An example can be found in Appendix B within the function 'TPOLL'. The syntax for s.in is <TIMED_BLOCK_HEAD> ::= IN <PERIOD> and for s.ni is <TIMED BLOCK>::= <TIMED_BLOCK_HEAD> DO <STMT GP> END IN.

These primitives are in the same format as they originally appeared in Ross's 8080 Realization Volume. They were added to the Z-80 Volume for purposes of completeness only and cannot be used in their present form. This is because the mechanism by which the CSDE program would implement a nested timing requirement is not functional. No effort was expended to correct this problem because it was reasoned that if a designer had an inclination to specify a timed block he could just as easily take the actions in question and put them in a separate task or function.

8. S.Stboolwait and s.boolwait

These primitives are generated when a boolean wait construct is specified. This construct would be specified by a designer when he wanted to check the results of an arithmetic expression for a fixed time period. The

expression is checked until either the result is true or the specified time has expired after which the rest of the task is executed. An example can be found in Appendix B in the task, 'MSGDSPLY'. The syntax for 's.stboolwait' is <WAIT HEAD> ::= WAIT UNTIL and for 's.boolwait' is <WAIT UNTIL> ::= <WAIT_HEAD> <EXPRESSION> : <PERIOD>.

The logic of the realization is fairly simple. The primitive, 's.stboolwait', sets the time to check the expression by calling another primitive, 's.setime'. This is analogous to setting a timer, in this case the counter timer chip (ctc) on the Pro-log cpu board. Then it establishes a label for the top of a mini polling routine. The expression primitive would appear between 's.stboolwait' and 's.boolwait'. Finally, 's.boolwait' completes the polling loop by first checking the result of the expression primitive. If the result is true the routine is exited. If the result is false, the current time is read from the ctc. If the time is expired the routine is exited. Otherwise a jump is executed to the top of the loop.

The primitive, 's.setime', is not generated by the Translator. It was written as a separate primitive to maintain the modularity of the Realization Volume and to allow testing of the CSDE 'call' instruction in the primitive, 's.stboolwait'. It sets up the ctc as a down counter which decrements at 1 ms intervals. Details of

available within the primitive. Consequently, the primitive realization was constructed to accept the following format:

s.forcons (COUNT,@C02,@C04,@03,@04,120:8,8,8).

The max loop count value also appears in the 's.forend' primitive as generated by the Translator. It is not required for proper construction of a for-loop and is ignored. Manual editing of the primitive list output file from the Translator is required whenever the for-loop primitives appear. Otherwise a fatal error will result upon running the CSDE program.

6. S.whilestart

This primitive is used in conjunction with the primitives, 's.whilecon' and 's.whend' to construct a while-do-loop. Its functions are to establish a label for the beginning of a while-do-loop and manipulate the max loop count using the global variable, 'reps'. This is similar to the method used in 's.forcons'. The CSDL syntax is '<WHILE> ::= WHILE'. Again, there is a problem as to where the max loop count is placed.

The CSDL syntax currently calls for max loop count to appear in the 's.whilecon' primitive and not in 's.whilestart'. This is wrong because the condition to be checked to determine if the while-do-loop should be continued appears as other primitives between 's.whilestart' and 's.whilecon'. Thus, these primitives, as well as the primitives appearing between 's.whilecon' and 's.whend', are

Thus, there would be no reason to incorporate a polling routine, knowing the result could never change. The CSDL syntax and the Translator should be corrected to allow additional primitives within the boolean wait routine such as 's.sensecond'. It would then be possible for the result of the final expression to change during the specified time period. The boolean wait realizations have been written accordingly.

9. S.waitleast

This primitive generates a software delay that is computed by multiplying the integer result of an arithmetic expression by a specified time period. The integer result is passed to 's.waitleast' from primitives that appear above it. The specified time period appears in 's.waitleast'. The syntax for it is <WAIT>::= WAIT <EXPRESSION> : <PERIOD>. An example can be found in Appendix B in the task, MSGDSPLY.

When code from this primitive is incorporated into a task the execution time of that task becomes variable based on internally computed results when the controller program is running. Since these changes in execution time occur completely external to CSDE, there is no way to achieve accurate timing statistics when the program is generated unless the designer can specify some maximum delay in the CSDL problem. Additionally, since the integer result passed to 's.waitleast' is passed via a variable and not an absolute number, the loop structure used in the

's.waitleast' realization required several additional arguments to be present in the title line than those generated by the Translator. The differences are displayed in Figure 8 as well as proposed CSDL syntax changes. Note that editing the primitive list requires more than just changing the 's.waitleast' title line. An additional variable may need to be added with an 's.var' primitive to

```
                    Current CSDL Syntax:

                  "WAIT SABLE+1: 500MS"

                    Proposed CSDL Syntax:

            "WAIT SABLE+1: 500MS: 1500MS"
                  -------      ^       ^
                     ^         :       :__ max time
        integer result__:      :_ period
        from this
        expression
```

Sample Output From Translator (unedited)

```
P  47s.waitleast (@TO1,8:500)
```

Required Format for Input to CSDE Program
(with argument explanations)

```
                  1    2    3   4   5    6

P  47s.waitleast (@TO2,@TO1,@05,@06,500,1500:8,8)

        1 - variable to be used for loop counter
        2 - variable containing integer result
        3 - top label
        4 - bottom label
        5 - time period
        6 - max allowed time period
```

Figure 8 Changes Required for 'S.waitleast'

implementation can be found in the comment lines for 's.setime' and 'h.clock' in Appendix E.

The CSDL syntax currently calls for the time period to appear in 's.boolwait' and not in 's.stboolwait'. This is wrong because CSDE needs the time period at the beginning of the boolean wait construct to insure accurate timing statistics are kept. This is accomplished by use of an attribute line in 's.stboolwait' as is similarly done in the primitive, 's.fixedwait'. Editing of the primitive list output from the Translator is required as shown in Figure 7.

Sample Output From the Translator (unedited)

```
P  36s.stboolwait(@03:)
P  37s.eq         (@T01,LIGHT,@C06:8,8,8)
p  38s.boolwait   (@T01,@03,@04:8,1700)     <-- 1700 is period
                                                in MS
```

Required Format for Input to CSDE Program

```
P  36s.stboolwait(@03,1700:)                <-- time period
P  37s.eq         (@T01,LIGHT,@C06:8,8,8)       here
P  38s.boolwait   (@T01,@03,@04:8)
```

Figure 7 Changes Required For Boolean Wait Primitives

A logical error exits in CSDL relating to the expression primitives that may appear between 's.stboolwait' and 's.boolwait'. Currently, only one expression may appear which means that once the boolean wait routine is entered there is no way that the result of the expression could change while waiting for the specified period to expire.

7. S.in and S.ni

These primitives are produced when a timed block is specified. A timed block is a nested set of actions within a task or function, with its own timing criteria. This timing criteria is in addition to any criteria specified for the parent task or function in the application timing table. An example can be found in Appendix B within the function 'TPOLL'. The syntax for s.in is <TIMED_BLOCK_HEAD> ::= IN <PERIOD> and for s.ni is <TIMED BLOCK>::= <TIMED_BLOCK_HEAD> DO <STMT GP> END IN.

These primitives are in the same format as they originally appeared in Ross's 8080 Realization Volume. They were added to the Z-80 Volume for purposes of completeness only and cannot be used in their present form. This is because the mechanism by which the CSDE program would implement a nested timing requirement is not functional. No effort was expended to correct this problem because it was reasoned that if a designer had an inclination to specify a timed block he could just as easily take the actions in question and put them in a separate task or function.

8. S.Stboolwait and s.boolwait

These primitives are generated when a boolean wait construct is specified. This construct would be specified by a designer when he wanted to check the results of an arithmetic expression for a fixed time period. The

serve as a counter for the loop structure produced by 's.waitleast'. Also, care must be taken when picking the values to be inserted in the label arguments so that they are different from label names used elsewhere in the primitive list.

10. Changes to Smith's Primitives

Three of the primitives retained from Smith's Z-80 Realization Volume were revised significantly. Many of the others required minor corrections, most regarding incorrect byte counts. In a few cases, comment lines were added to clarify certain points regarding the structure of a particular primitive. Generally, Smith was thorough and the logic of his Z-80 assembly code routines was flawless.

The primitive, 's.main', appears in the second line of every primitive list generated by the Translator. The 's.main' realization contains the code that appears at the top of every software output from CSDE. Smith included statements that would allow a designer to specify a debug mode. A controller program produced from CSDE in debug mode could be run and tested on a C/PM based microcomputer. These references to debug mode were eliminated since, with the use of the CSDE workstation, testing on the Pro-log is more effective. Another capability of 's.main' was retained to generate blocks of code within a controller program that are only executed once before the monitor loop is entered. Smith referred to this as hardware initializations. Although

not implemented in the revised Realization Volume, such blocks could be realized by modifying existing primitive realizations with 'incl' or 'call' statements. These statements could, in turn, reference other software primitives whose code would be executed only once upon

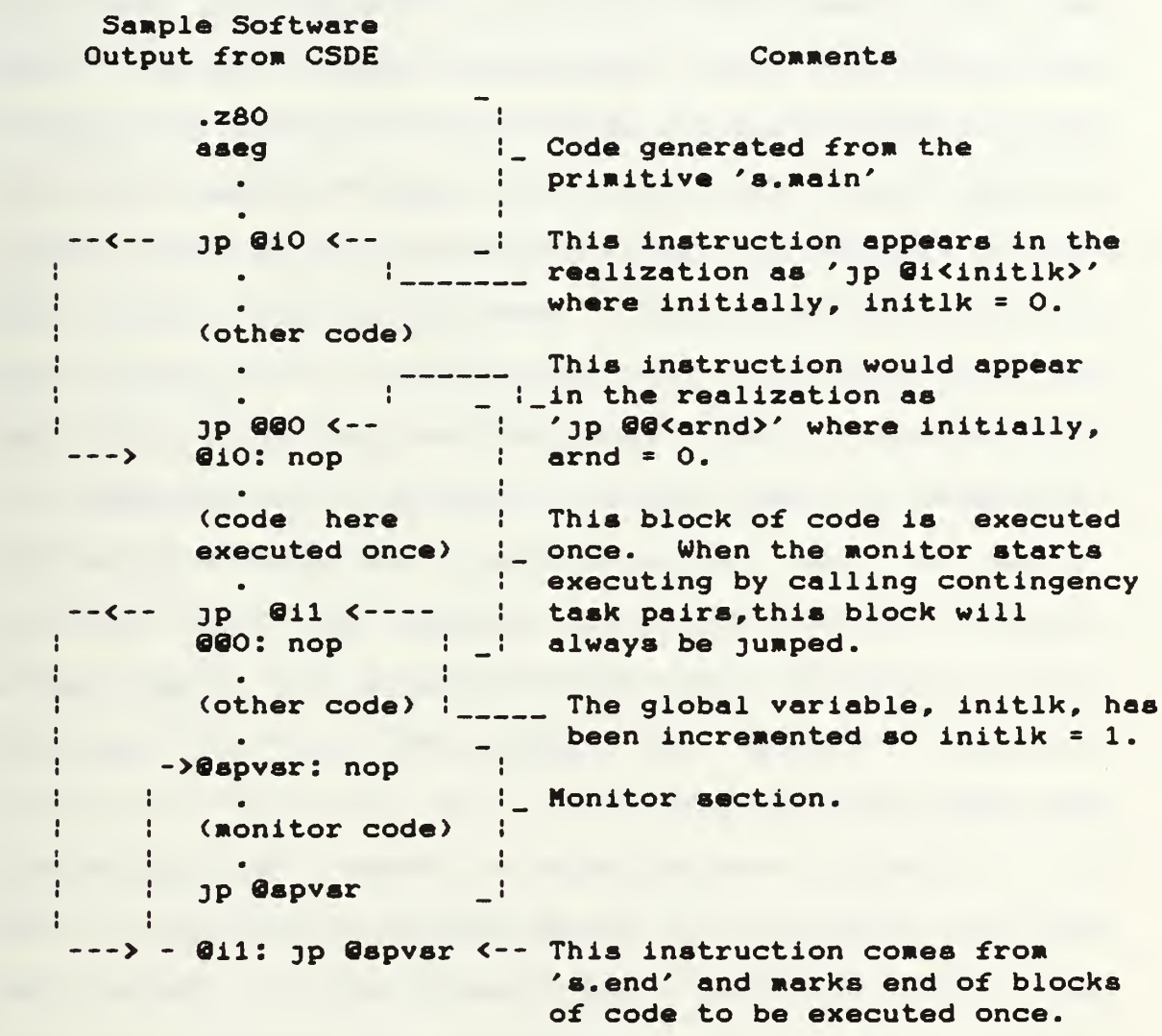


Figure 9 Realization of Blocks of Code to be Executed Once Only

inclusion in a controller program. This feature is available for use, utilizing the global variables 'initlk' and 'arnd'. Figure 9 demonstrates how blocks of code to be executed once (for initializations) might be realized.

The primitive, 's.monitor', also contained a means of generating code within a controller program that would only be executed once before entering the monitor section. This was different from the method use in 's.main' in that a boolean flag, '@initial', was checked to cause a jump to another segment of code that might be used for such things as initializing variables. This initializing block of code was realized from two primitives, 's.initalcons' and 's.initalend'. These primitives are not produced by the Translator and were discarded along with the statements in 's.monitor', that referred to them. The method available in 's.main' for initializations is simple and more flexible. Initial values of variables are currently set at zero when a controller program is assembled by use of assembler statements such as 'defw 0'.

Finally, the primitive, 'h.clock', was rewritten to correctly describe all jumper connections required on the cpu board to implement a 1 millisecond clock. The original version of 'h.clock' listed only one of the six connections required to enable channel 0 to feed channel 1 of the ctc.

# IV. TESTING AND DEMONSTRATION

The objective during testing was to verify that every primitive in the revised Z-80 Realization Volume could be processed through CSDE to produce error free code that would run on the Pro-Log computer. This applied to newly constructed primitive realizations as well as those retained from Smith' Z-80 Volume. All primitives were successfully tested at least once except for 's.in', 's.ni', and 's.not'. The first two were discussed in section III.C.7. and the latter was not tested because the Translator would not generate it. Many unsuccessful attempts were made to discover the proper syntax for 's.not' since the Backus-Naur form specified by Carson failed to work.

To minimize the potential for multiple errors within one test problem only one new primitive was tested at a time. This involved writing simple CSDL problems for each primitive usually with only two contingency task pairs. The testing was not exhaustive. For example, if a primitive like 's.ge' (checks condition for greater than or equal to) was tested, not all possible combinations of positive and negative numbers were submitted as input data. This was because of time constraints and the fact that Smith already tested the logic of his Z-80 code during development of the original Z-80 Realization Volume.

As discussed below in section IV.C., the CSDE program underwent extensive revision to improve efficiency and ease of use. An indirect benefit of primitive testing for this thesis was that the revised CSDE program was able to be debugged. While important to the overall CSDE project this sometimes caused great frustration when CSDE program errors slowed down the testing process. Primitive testing also validated the usefulness of the CSDE workstation.

## A. TESTING SEQUENCE AND EXAMPLE

A detailed user's manual for working within the CSDE environment is contained in Poole's thesis [Ref. 12]. It specifies the exact command sequences to use the CSDE program on the NPGS Computer Science VMS Vax 11/780. It also explains how to use the CSDE workstation to transfer files between the Z-100 and the VAX, assemble and link Z-80 programs produced by CSDE, and download resultant hex files to the Pro-Log. The process is summarized as follows.

Once a new primitive realization was written and added to the Realization Volume as discussed in Chapter 3, a short CSDL problem was written to exercise it. Appendix B shows the proper CSDL format to generate every primitive except 's.not'. The file containing the CSDL problem was renamed DAT.DAT, as required for input to the Translator. The problem was then run through the Translator to generate the primitive list, application timing table, symbol table, and

Translator error file. These files were discussed in Chapter 2. It was important to view the Translator error file before inputting the problem to CSDE because this was the only way to know if the CSDL program contained any syntax errors. Any manual changes to the primitive list were made using the EDT editor available on the VAX. Next, the CSDE program, currently named CLIB, was run to produce a software listing, a hardware listing and a debug file. Different levels of debugging may be selected from the initial CLIB menu. The software listing was then downloaded to the Z-100 microcomputer, part of the CSDE workstation. On the Z-100, it was assembled and linked to produce a hex file. The hex file was downloaded from the Z-100 to the Pro-Log where it was finally executed.

As mentioned above, different levels of debugging may be selected when running the CSDE program. When level 0 is selected only the actual error message lines will appear in the debug file. When level 3 is selected an extensive chronological record of CSDE program execution is written to the debug file. In most cases it was easiest to select level 0 and if errors developed, rerun the problem with a more detailed debug level selected. Another point is that just because the CSDE program flags errors does not mean that an unsatisfactory realization has been produced. In some cases, such as with 's.fixedwait', only non fatal

informative errors are generated. This was mentioned in section III.C.2.

An actual test program and all related files are contained in Appendix F. This CSDL problem was written to test the primitives associated with a while-do loop. Referring to the problem, 'FUNCTION EACH1' is a contingency which senses an input value, stores it in the variable, 'ARG1', and sets the boolean variable, 'EACH1', equal to -1 if 'ARG1' is less than or equal to 2. The boolean variable is set equal to -1 because -1 decimal is represented by FF hexadecimal in twos complement form. A boolean true value is defined as FF hex. The net result is that if 'ARG1' is less than or equal to 2 than the contingency is true. The 'CONTINGENCY LIST' specifies that if 'EACH1' is true then 'TASK ONLITA' must be executed. Both the contingency and task must be completed within 1600 milliseconds including any other blocks of code that are executed during the remaining portion of the current monitor cycle.

The while-do loop comprises the bulk of 'TASK ONLITA'. The net result of the while-do loop is that the values 1, 3, 5, and 7 will be output at 250 ms intervals and can be viewed in binary form on the 8 leds of the Pro-log keyboard/display card. This display will only occur if the contingency is true, i.e. both keyboard/display card rocker switches are off or only the right switch is on.

The second contingency task pair, 'EACH5' and 'OFFLT', causes a 500 ms delay with all leds off for any of the four possible input values. See the discussion of 's.sensecond' in section III.C.1. for more information on the four possible input values.

Following the CSDL problem are the three Translator output listings used by the CSDE program. They are the primitive list, application timing file and symbol table. The primitive list as shown in Appendix F, was modified from the original Translator output to position the value for max loop count as the second argument in the primitive 's.whilestart'. See Figure 6 in section III.C.6 for an illustration of this change.

Finally, the software, hardware and debug listings are displayed exactly as produced from the CSDE program. The software listing is ready to be assembled and the debug listing was generated in debug level 0. The errors contained therein are for information only and relate to the missing colons in the 's.fixedwait' primitives.

## B. TRANSLATOR ERRORS

During the course of primitive testing, some errors were discovered in the format of primitives generated by the Translator. Other Translator errors relate to the manner in which it handles numbers and determines the precision of

internally generated variables. All Translator errors are summarized in Appendix G.

Most format errors relate to the placement of values for loop counts or time within the primitive title lines. These have been documented in Chapter III.C. and apply to the following primitives: 's.forcons', 's.whilestart', 's.whilecon'', 's.stboolwait', 's.boolwait', and 's.waitleast'. The primitive realization for 's.waitleast' also contains many more arguments than in the Translator version and modification to its syntax in CSDL is also required. This is detailed in section III.C.9. A final format error is that the primitive, 's.fixedwait', is generated without the required colon after the value for time. The functional mapper module of the CSDE program has been modified to accommodate this error. When encountered by the CSDE program, error messages are generated and then the required colon is inserted in the correct location. All errors except for 's.fixedwait' require manual correction by editing the primitive list prior to running the CSDE program.

Although CSDL syntax rules allow time units as small as nanoseconds, the Translator correctly generates only time values accurate to the next lowest millisecond. For example, if a CSDL problem contained the statement, 'WAIT 600 US', the Translator would generate the primitive, 's.fixedwait (0)'. All time values generated by the

Translator are in milliseconds and any primitive realizations that take input values of time must be written accordingly. Milliseconds are excellent time units when long delays are required, especially for demonstration programs that utilize the leds on the Pro-log keyboard/display board. However, for more flexibility in possible controller designs, the shorter time units should be available for use. This is because some applications, for example a jet engine start controller, might require more stringent response times.

There are two other problems relating to the Translator's handling of numbers in general. One is that it only recognizes integers. For example, if a CSDL problem segment was written as 'COUNT:=COUNT+10.6', the number, 10.6, would be passed to the primitive list as simply 10. The second problem involves the criteria used to create 16 bit constants instead of 8 bit constants. For example, if a CSDL problem segment was written as 'COUNT:=128', the primitive, 's.cons (@C01,128:8)' would appear in the primitive list. If the value was 129 instead of 128, the primitive, 's.cons (@C01,129:16)' would appear. This is wrong because the largest positive twos complement number that can be specified in an 8 bit word is 127. Therefore the decision point for specifying 8 bit or 16 bit constants should be between 127 and 128, not 128 and 129.

One other error relating to the precision of variables occurs when dealing with primitives that use boolean variables. For example, the primitive, 's.eq', has three arguments and compares the values of the second and third arguments for equality. Upon completion of the equality test, the first argument is set equal to FF hex for true or 00 hex for false. Since the first argument is always used as a boolean variable, an 8 bit precision will always be sufficient even if the other arguments call for 16 bit precision. All relational primitive realizations were written assuming the boolean argument will always have an 8 bit precision. Unfortunately, the Translator generates a 16 bit boolean argument whenever either of the other arguments

Example Relational Primitive (unedited)

```
P  30s.eq        (T@11,ARG1,CONST:16,16,16)
                    ^
                    |_  16 bit variable generated
                        by Translator to pass
                        boolean result;  only needs
                        to have 8 bit precision
```

Required Format for Input to CSDE Program

```
P  30s.eq        (T@01,ARG1,CONST:8,16,16)
```

Figure 10 Changes Required for Relational Primitives

has a 16 bit precision. This results in criteria check errors from the CSDE program whenever large numbers

requiring 16 bit precision are compared within a relational primitive. Primitive lists containing such errors must be corrected before running the CSDE program. An example is contained in Figure 10.

The final Translator error requiring correction is that if variables are declared as type DUPLEX, any subsequent use of those variables in a 'SENSE ', or 'ISSUE' statement results in syntax error messages. This is wrong because the whole concept of DUPLEX type variables involves their use for either input or output. This was discussed in section III.C.1.

## C. THE CSDE PROGRAM

As mentioned in Chapter I, the CSDE program underwent revision during the course of research for this thesis. Most changes are transparent to the user and involved streamlining the CSDE program source code to improve efficiency. Additional improvements over the NEWCSDL version used by Smith and Riley include the addition of a user friendly menu and elimination of the need for the input file, MONTER.DAT. This file contained the primitives required to generate the monitor section of a controller program. The monitor primitives were already contained in the Realization Volume and thus, MONTER.DAT was really not needed.

The revised CSDE program, CLIB, was exercised frequently while developing and testing new primitive realizations. Many errors that had been introduced during its revision were identified and corrected as testing progressed. The importance of good communication between the user (me) and software maintenance personnel (Mr. Lukas) was made very clear. Despite delays due to errors in CLIB, testing for this thesis could not have been completed without it. The CSDE workstation also proved invaluable in reducing testing time per primitive compared to the methods used by Smith.

Realization testing and debugging CLIB uncovered an important idiosyncrasy of CSDE. Specifically, the input file containing the list of global variables, usually named GLOBALS.DAT, has a strict format. Certain positions within the globals file are reserved for global variables used internally by CLIB. If a new global variable used within a primitive realization is accidentally placed in one of these 'hard wired' positions unpredictable errors will be generated. The current global variable file contains some global variables that are not found in the Revised Z-80 Realization Volume. These variables were added by Smith because they were used in some of his primitive realizations from the original Z-80 Realization Volume. Since these primitives have been deleted from the Revised Z-80 Realization Volume, some global variables added by Smith serve no function as far as primitives are concerned.

However, because of their position in GLOBALS.DAT they might still be used internally by CLIB. Thus, global variables not found in the Revised Realization Volume have been retained to insure that CLIB runs correctly. If any new global variables are added in the future they should be added at the bottom of the file. Also, the number at the top of the file, indicating the total number of global variables, should be adjusted accordingly. Figure 11 displays the contents of the current global variable file.

| Contents of Global Variable File | Applicable Notes |
|---|---|
| 022 | |
| arnd 0. | 1,2,3 |
| bdos 5. | 2,3 |
| chips 0. | 2,4 |
| clock 0. | 5 |
| initlk0. | 1,3,5 |
| reps 1. | 4,5,6 |
| natode0. | 2,3 |
| natodp0. | 2,3,6 |
| ndtoae0. | 2,3 |
| ndtoap0. | 2,3 |
| ninout0. | 2,3 |
| nkey 0. | 2,3 |
| nled -1. | 2,3 |
| nodgt 0. | 2,3 |
| norom 0. | 2,3 |
| nrockr0. | 2,3 |
| ramptr0. | 3,5 |
| romptr0. | 3,5 |
| scrtch0. | 3,5 |
| slot 0. | 3,5 |
| keybrd0. | 3,5 |
| tmblck0. | 3,5 |

Note Explanations

1 -- Available for use if initialization primitives are added. See Figure 9.

2 -- Not used in Revised Z-80 Realization Volume.

3 -- Added by Smith, used in original Z-80 Realization Volume.

4 -- Used internally by the CSDE program, CLIB.

5 -- Used in Revised Z-80 Realization Volume.

6 -- Known 'hard wired' position used by CLIB.

Figure 11 Contents of Global Variable File

# V. CONCLUSIONS

The goals for this thesis have been accomplished. All but a few discrepancies have been resolved between the Translator and the Z-80 Realization Volume. Numerous test problems have been run through the entire CSDE system, from CSDL problem statement to operating program on the Pro-log microcomputer. These test problems, as implemented on the Pro-log, can be considered true controller realizations since changes in input values result in different output values.

More complex problems should be demonstrated in the future. This will require the addition of more complex I/O hardware with accompanying modification to the I/O primitive realizations. If, for example, Riley's jet engine controller were to be demonstrated, a means would also be needed of simulating the various parameters to be sensed. An array of potentiometers connected to analog to digital converters might be utilized for this purpose. Similarly, outputs from the controller would need to be displayed differently than the current method using leds. Complex problems might require more memory than the currently available 16k RAM.

The Translator errors discussed in chapter IV should be corrected. This would result in complete compatibility

between component parts of CSDE and would eliminate the need for manual modification of intermediate primitive lists. Another modification to the Translator might be to improve the clarity of its error messages when CSDL syntax errors are encountered. Currently, these messages are extremely difficult to understand since they only point out the location of a syntax error within a CSDL problem. Until a syntax directed editor or language manual is developed for CSDL, the error detection facility of the Translator is the only aid available for writing correct CSDL problems.

This thesis research was primarily an exercise in software maintenance. The problems encountered and effort expended to solve those problems were invaluable in demonstrating why software maintenance requires such large proportions of government and industry data processing resources. For example, the concept of continuity between development personnel became quite clear as many questions arose concerning previous researchers work. Had personnel such as Carson, Smith, and Riley been available for consultation, much time probably could have been saved. The importance of good communication between users and maintenance personnel was proven as the revised CSDE program was successfully debugged. This also illustrated the difficulties that arise when two components of a system that affect each other undergo maintenance at the same time. Errors in the revised CSDE program caused some unanticipated

delays in the testing of new primitive realizations. Finally, good documentation is mandatory for a successful software maintenance project.

## APPENDIX A

## PRIMITIVES AVAILABLE FROM TRANSLATOR

This appendix contains a list of all primititives that can be produced by Carson's translator. The corresponding production number can be used to find the Backus-Naur syntax structure in the listing of the CSDL language in Carson's thesis [Ref 8: pp. 47-54]. A brief phrase describing each primitive is also supplied.

| PRIMITIVE | PROD NUMBER | MEANING |
|---|---|---|
| ADD | 23 | Addition |
| SUB | 21 | Subtraction |
| MULT | 23 | Multiply |
| DIVIDE | 23 | Divide |
| LT | 25 | Less Than |
| LE | 25 | Less Than or Equal To |
| EQ | 25 | Equal To |
| GT | 25 | Greater Than |
| GE | 25 | Greater Than of Equal To |
| NE | 25 | Not Equal To |
| NOT* | 22 | Boolean Not |
| AND | 27 | Boolean And |
| OR | 29 | Boolean Or |
| IMPLICATE | 31 | Logical Implication |
| EQUIVALENC | 33 | Logical Equivalence |
| LOC | 37 | Location in IF THEN |
| JMPF | 38 | Jump If False |
| WHEND | 39 | End of WHILE Construct |
| WHILECON | 40 | Test Portion of WHILE Construct |

| | | |
|---|---|---|
| WHILESTART | 41 | Beginning of WHILE Construct |
| FOREND | 42 | End of FOR LOOP |
| FORCONS | 43 | Condition Tested in FOR LOOP |
| CALL | 44 | Generate a Pocedure Call |
| ASSIGN | 49 | Assignment Statement |
| SENSECOND | 50 | Sense a Condition for Data Input |
| ISSUEVENT | 51 | Data Output |
| NI | 61 | End of IN Construct(Timed Block) |
| IN | 62 | Beginning of IN Construct |
| FIXEDWAIT | 63 | Timed Software Delay |
| WAITLEAST | 64 | Minimum Wait |
| BOOLWAIT | 65 | Body of Boolean Wait Constuct |
| STBOOLWAIT | 66 | Wait Until (Start of Boolean Wait Construct) |
| INPUTPORT | 86 | Input Specification |
| OUTPUTPORT | 87 | Output Specification |
| IN/OUTPORT | 94 | Duplex (input or output) Spec |
| VAR | 105 | Variable Assignment |
| EXITPROC | 146 | Marks Exit of Procedure, Function, or Task |
| PROC | 145 | Marks Beginning of Procedure, Function, or Task |
| SYSTEM | 190 | Generates System Title |
| @ | 102 | Generated by Translator as Location Assignment Place Holders |
| CONS | 190 | Constant |

* Although the Translator is supposed to be able to

generate a 'NOT' primitive, attempts to do so were unsuccessful.

## APPENDIX B

## CSDL TEST PROGRAM

This appendix contains Carson's CSDL test program to exercise the Translator to produce all possible primitives. As originally written, it did not contain the CSDL structures to produce the primitives, 's.aub', 's.not', 's.call', and 's.waitleast'. Structures have been added to produce all except 's.not'. Also, the structure, 'DO MANUAL 4;', originally found in the contingency list, caused the Translator to produce an error message even though it appeared to be correct according to CSDL. This structure was deleted. The resulting primitive list and application timing table are contained in Appendix C.

```
      DESIGNER : "HILL CARSON/ MODIFIED BY BOB VOGEL"
      DATE : "05-31-84/02-20-85"
      PROJECT :"TEST PROGRAM TO EXERCISE TRANSLATOR"


DESIGN CRITERIA
       METRIC FIRST;
       VOLUMES 1;
       MONITORS 1;


ENVIRONMENT

INPUT:KEYFLG,1,TTL; KEYCHAR,8,TTL;
      ACNUM,8,TTL;
END INPUT;

OUTPUT: MENU,8,TTL;POLL,8,TTL; END OUTPUT;

DUPLEX MSGVDT,8,TTL; END DUPLEX;

ARITHMETIC: KEYINMAIN,8; MINTAC,8; MMSGDSPLY,8;
      AC0,8; AC1,8; AC2,8; AC3,8; AC4,8;
      INTPERIOD,8; MSG0,8;
      MSG1,8; MSG2,8;
      NEXTAC,8; TPOLL,1; COUNT,8;
END ARITHMETIC;
```

```
PROCEDURES


FUNCTION KEYINMAIN:
        BINARY,1;
        SENSE (KEYFLG);
        IF KEYFLG=1 THEN KEYINMAIN:=1; END IF;
END KEYINMAIN;

TASK KBINPMAIN;
        MENU:=0; ISSUE (MENU);
        SENSE (KEYCHAR);
        IF KEYCHAR=1 THEN MINTAC :=1; END IF;
        WAIT 10MS;
END KBINPMAIN;



TASK MANUAL;
        IF AC0/=0 THEN POLL:=0; ISSUE (POLL); END IF;
        IF AC1<0 THEN POLL:=1; ISSUE (POLL); END IF;
        IF AC2<=0 THEN POLL:=2; ISSUE (POLL); END IF;
        IF AC3>=0 THEN POLL:=3; ISSUE (POLL); END IF;
        IF AC4>0 THEN POLL:=4; ISSUE (POLL); END IF;
END MANUAL;



FUNCTION TPOLL:
      BINARY,1;
      IF INTPERIOD=30 THEN IN 30 M DO TPOLL:=1; END IN;
      END IF;
END TPOLL;


TASK MSGDSPLY;
        KBINPMAIN;
        MMSGDSPLY:=0;
        WAIT MMSGDSPLY+1: 500MS;
        WAIT UNTIL MMSGDSPLY = 0: 10MS;
END MSGDSPLY;



TASK LOGIN;
        ACNUM:=0;
        FOR COUNT FROM 1 TO 4:4 DO
              SENSE (KEYCHAR);
              ACNUM:=ACNUM-5;
              ACNUM:=(ACNUM*10)+KEYCHAR;
        END FOR;
```

```
        IF NEXTAC=0 AND AC0=0 THEN AC0:=ACNUM; END IF;
        IF NEXTAC=1 OR AC1=0 THEN AC1:=ACNUM; END IF;
        IF NEXTAC=2 => AC2=0 THEN AC2:=ACNUM; END IF;
        IF NEXTAC=3 == AC3=0 THEN AC3:=ACNUM; END IF;
        WHILE ACNUM = 1 : 4 DO
              AC4 := 4;
              ACNUM := ACNUM + 1;
        END WHILE;
       END LOGIN;



TASK LOGOUT;
        ACNUM:=0;
        FOR COUNT FROM 1 TO 4:4 DO
             SENSE (KEYCHAR);
             ACNUM:=(ACNUM*10)/KEYCHAR;
        END FOR;
        IF AC0=ACNUM THEN AC0:=0; END IF;
END LOGOUT;


CONTINGENCY LIST
        WHEN KEYINMAIN : 100 MS DO KBINPMAIN;
        EVERY 20MS DO KBINPMAIN;
        AT 300MS DO KBINPMAIN;
END
```

## APPENDIX C

### PRIMITIVE LIST, APPLICATION TIMING TABLE AND SYMBOL TABLE PRODUCED BY TRANSLATOR FROM CSDL TEST PROGRAM

This appendix contains the unedited output from the Translator that results from the CSDL test program in Appendix B. The first item is the primitive list that comes out in the file PRIMFILE.DAT, the second item is the application timing table that comes out in the file IADEFL.DAT, and the last item is the symbol table that comes out in the file, SYMFILE.DAT.

### PRIMFILE.DAT

```
P   2s.MAIN        (::)
P   3d:FIRST       :          1:          1:
P   4s.inputport   (KEYFLG,TTL:1)
P   5s.inputport   (KEYCHAR,TTL:8)
P   6s.inputport   (ACNUM,TTL:8)
P   7s.outputport  (MENU,TTL:8)
P   8s.outputport  (POLL,TTL:8)
P   9s.in/outport  (MSGVDT,TTL:8)
P  10s.var         (KEYINMAIN:8,0)
P  11s.var         (MINTAC:8,0)
P  12s.var         (MMSGDSPLY:8,0)
P  13s.var         (AC0:8,0)
P  14s.var         (AC1:8,0)
P  15s.var         (AC2:8,0)
P  16s.var         (AC3:8,0)
P  17s.var         (AC4:8,0)
P  18s.var         (INTPERIOD:8,0)
P  19s.var         (MSG0:8,0)
P  20s.var         (MSG1:8,0)
P  21s.var         (MSG2:8,0)
P  22s.var         (NEXTAC:8,0)
P  23s.var         (TPOLL:1,0)
P  24s.var         (COUNT:8,0)
P  25t.generated   for: KEYINMAIN              ******************
P  26s.proc        (KEYINMAIN:)
P  27s.sensecond   (KEYFLG:1)
P  28s.eq          (@T01,KEYFLG,@C01:8,1,8)
P  29s.jmpf        (@T01,@01:8)
P  30s.assign      (KEYINMAIN,@C01:1,8)
P  31s.loc         (@01:)
P  32s.exitproc    (KEYINMAIN:)
P  33t.generated   for: KBINPMAIN              ******************
P  34s.proc        (KBINPMAIN:)
P  35s.assign      (MENU,@C02:8,8)
P  36s.issuevent   (MENU:8)
```

```
P  37s.sensecond (KEYCHAR:8)
P  38s.eq        (@T01,KEYCHAR,@C01:8,8,8)
P  39s.jmpf      (@T01,@02:8)
P  40s.assign    (MINTAC,@C01:8,8)
P  41s.loc       (@02:)
P  42s.fixedwait (10)
P  43s.exitproc  (KBINPMAIN:)
P  44t.generated for: MANUAL                ****************
P  45s.proc      (MANUAL:)
P  46s.ne        (@T01,AC0,@C02:8,8,8)
P  47s.jmpf      (@T01,@03:8)
P  48s.assign    (POLL,@C02:8,8)
P  49s.issuevent (POLL:8)
P  50s.loc       (@03:)
P  51s.lt        (@T01,AC1,@C02:8,8,8)
P  52s.jmpf      (@T01,@04:8)
P  53s.assign    (POLL,@C01:8,8)
P  54s.issuevent (POLL:8)
P  55s.loc       (@04:)
P  56s.le        (@T01,AC2,@C02:8,8,8)
P  57s.jmpf      (@T01,@05:8)
P  58s.assign    (POLL,@C03:8,8)
P  59s.issuevent (POLL:8)
P  60s.loc       (@05:)
P  61s.ge        (@T01,AC3,@C02:8,8,8)
P  62s.jmpf      (@T01,@06:8)
P  63s.assign    (POLL,@C04:8,8)
P  64s.issuevent (POLL:8)
P  65s.loc       (@06:)
P  66s.gt        (@T01,AC4,@C02:8,8,8)
P  67s.jmpf      (@T01,@07:8)
P  68s.assign    (POLL,@C05:8,8)
P  69s.issuevent (POLL:8)
P  70s.loc       (@07:)
P  71s.exitproc  (MANUAL:)
P  72t.generated for: TPOLL                 ****************
P  73s.proc      (TPOLL:)
P  74s.eq        (@T01,INTPERIOD,@C06:8,8,8)
P  75s.jmpf      (@T01,@08:8)
P  76s.in        (1800000)
P  77s.assign    (TPOLL,@C01:1,8)
P  78s.ni        (::)
P  79s.loc       (@08:)
P  80s.exitproc  (TPOLL:)
P  81t.generated for: MSGDSPLY              ****************
P  82s.proc      (MSGDSPLY:)
P  83s. call     (KBINPMAIN:)
P  84s.assign    (MMSGDSPLY,@C02:8,8)
P  85s.add       (@T01,MMSGDSPLY,@C01:8,8,8)
P  86s.waitleast (@T01,8:500)
P  87s.atboolwait(@09:)
```

```
P  88s.eq          (@TO1,MMSGDSPLY,@CO2:8,8,8)
P  89s.boolwait    (@TO1,@09,@10:8,10)
P  90s.exitproc    (MSGDSPLY:)
P  91t.generated for: LOGIN                    ****************
P  92s.proc        (LOGIN:)
P  93s.assign      (ACNUM,@CO2:8,8)
P  94s.forcons     (COUNT,@CO1,@CO5,@11,@12:8,8,8,4)
P  95s.sensecond   (KEYCHAR:8)
P  96s.sub         (@TO1,ACNUM,@CO7:8,8,8)
P  97s.assign      (ACNUM,@TO1:8,8)
P  98s.mult        (@TO1,ACNUM,@CO8:8,8,8)
P  99s.add         (@TO1,@TO1,KEYCHAR:8,8,8)
P 100s.assign      (ACNUM,@TO1:8,8)
P 101s.forend      (COUNT,@11,@12:8,4)
P 102s.eq          (@TO1,NEXTAC,@CO2:8,8,8)
P 103s.eq          (@TO2,ACO,@CO2:8,8,8)
P 104s.and         (@TO1,@TO1,@TO2:8,8,8)
P 105s.jmpf        (@TO1,@13:8)
P 106s.assign      (ACO,ACNUM:8,8)
P 107s.loc         (@13:)
P 108s.eq          (@TO1,NEXTAC,@CO1:8,8,8)
P 109s.eq          (@TO2,AC1,@CO2:8,8,8)
P 110s.or          (@TO1,@TO1,@TO2:8,8,8)
P 111s.jmpf        (@TO1,@14:8)
P 112s.assign      (AC1,ACNUM:8,8)
P 113s.loc         (@14:)
P 114s.eq          (@TO1,NEXTAC,@CO3:8,8,8)
P 115s.eq          (@TO2,AC2,@CO2:8,8,8)
P 116s.implicate   (@TO1,@TO1,@TO2:8,8,8)
P 117s.jmpf        (@TO1,@15:8)
P 118s.assign      (AC2,ACNUM:8,8)
P 119s.loc         (@15:)
P 120s.eq          (@TO1,NEXTAC,@CO4:8,8,8)
P 121s.eq          (@TO2,AC3,@CO2:8,8,8)
P 122s.equivalenc(@TO1,@TO1,@TO2:8,8,8)
P 123s.jmpf        (@TO1,@16:8)
P 124s.assign      (AC3,ACNUM:8,8)
P 125s.loc         (@16:)
P 126s.whilestart(@17:)
P 127s.eq          (@TO1,ACNUM,@CO1:8,8,8)
P 128s.whilecon    (@TO1,@18:4)
P 129s.assign      (AC4,@CO5:8,8)
P 130s.add         (@TO1,ACNUM,@CO1:8,8,8)
P 131s.assign      (ACNUM,@TO1:8,8)
P 132s.whend       (@17,@18:)
P 133s.exitproc    (LOGIN:)
P 134t.generated for: LOGOUT                   ****************
P 135s.proc        (LOGOUT:)
P 136s.assign      (ACNUM,@CO2:8,8)
P 137s.forcons     (COUNT,@CO1,@CO5,@19,@20:8,8,8,4)
P 138s.sensecond   (KEYCHAR:8)
```

```
P 139s.mult       (@TO1,ACNUM,@CO8:8,8,8)
P 140s.divide     (@TO1,@TO1,KEYCHAR:8,8,8)
P 141s.assign     (ACNUM,@TO1:8,8)
P 142s.forend     (COUNT,@19,@20:8,4)
P 143s.eq         (@TO1,ACO,ACNUM:8,8,8)
P 144s.jmpf       (@TO1,@21:8)
P 145s.assign     (ACO,@CO2:8,8)
P 146s.loc        (@21:)
P 147s.exitproc   (LOGOUT:)
P 148t.generated for:   SYSTEM              ******************
P 149s.cons       (@CO1,1:8)
P 150s.cons       (@CO2,0:8)
P 151s.cons       (@CO3,2:8)
P 152s.cons       (@CO4,3:8)
P 153s.cons       (@CO5,4:8)
P 154s.cons       (@CO6,30:8)
P 155s.cons       (@CO7,5:8)
P 156s.cons       (@CO8,10:8)
P 157s.var        (@TO1:8)
P 158s.var        (@TO2:8)
```

IADEFL.DAT

```
A   1 :           :KBINPMAIN :MS: 100,   0,   0,   0,   0
A   2 :           :KBINPMAIN :MS:  20,   0,   0,   0,   0
A   3 :           :KBINPMAIN :MS: 300,   0,   0,   0,   0
```

SYMFILE.DAT

```
S. INPUTPORT(KEYFLG,TTL:1)
S. INPUTPORT(KEYCHAR,TTL:8)
S. INPUTPORT(ACNUM,TTL:8)
S.OUTPUTPORT(MENU,TTL:8)
S.OUTPUTPORT(POLL,TTL:8)
S.IN/OUTPORT(MSGVDT,TTL:8)
S.VARIABLE  (KEYINMAIN:8,0)
S.VARIABLE  (MINTAC:8,0)
S.VARIABLE  (MMSGDSPLY:8,0)
S.VARIABLE  (ACO:8,0)
S.VARIABLE  (AC1:8,0)
S.VARIABLE  (AC2:8,0)
S.VARIABLE  (AC3:8,0)
S.VARIABLE  (AC4:8,0)
S.VARIABLE  (INTPERIOD:8,0)
S.VARIABLE  (MSG0:8,0)
S.VARIABLE  (MSG1:8,0)
S.VARIABLE  (MSG2:8,0)
```

```
S.VARIABLE   (NEXTAC:8,0)
S.VARIABLE   (TPOLL:1,0)
S.VARIABLE   (COUNT:8,0)
S.LOC        (@01:)
S.LOC        (@02:)
S.LOC        (@03:)
S.LOC        (@04:)
S.LOC        (@05:)
S.LOC        (@06:)
S.LOC        (@07:)
S.LOC        (@08:)
S.LOC        (@13:)
S.LOC        (@14:)
S.LOC        (@15:)
S.LOC        (@16:)
S.LOC        (@21:)
S.CONS       (@C01:1,8)
S.CONS       (@C02:0,8)
S.CONS       (@C03:2,8)
S.CONS       (@C04:3,8)
S.CONS       (@C05:4,8)
S.CONS       (@C06:30,8)
S.CONS       (@C07:5,8)
S.CONS       (@C08:10,8)
```

# APPENDIX D

## COMPARISON OF PRIMITIVES

This appendix displays a comparison of the primitives available from the Revised Z-80 Realization Volume (Appendix E) and the primitives available from the Translator. Primitives from the Revised Volume are in the same format as they appear in the index in Appendix E except that the seven numeric values following the second column are not shown. Primitives from the Translator are in the same format as they appear in the primitive listing that is generated by the Translator when Carson's CSDL test program (Appendx C) is run through it. Differences in the arrangement of arguments between some Realization and Translator primitives are due errors in the Translator (summarized in Appendix G). The primitive, 's.in/outport', was not added to the Realization Volume.

| REVISED Z-80 VOLUME | | CARSON'S TRANSLATOR | |
|---|---|---|---|
| s.add | (rslt,arg1,arg2:0,8,0,8,0,8: ) | s.add | (@T01,@T01,KEYCHAR:8,8,8) |
| s.add | (rslt,arg1,arg2:0,16,0,16,0,16: ) | | |
| s.and | (rslt,arg1,arg2:0,8,0,8,0,8: ) | s.and | (@T01,@T01,@T02:8,8,8) |
| s.assign | (var,data:0,8,0,8: ) | s.assign | (KEYINMAIN,@C01:1,8) |
| s.assign | (var,data:0,16,0,16: ) | | |
| s.boolwait | (rslt,top,bot:0,8: ) | s.boolwait | (@T01,@09,@10:8,10) |
| s.call | (nam:: ) | s.call | (ONLITA:) |
| s.cons | (nam,val, :0,8: ) | s.cons | (@C01,1:8) |
| s.cons | (nam,val, :0,16: ) | | |
| s.divide | (rslt,arg1,arg2:0,8,0,8,0,8: ) | s.divide | (@T01,@T01,KEYCHAR:8,8,8) |
| s.divide | (rslt,arg1,arg2:0,16,0,16,0,16: ) | | |
| s.end | (:: )* | | |
| s.eq | (rslt,arg1,arg2:0,8,0,8,0,8: ) | s.eq | (@T01,KEYFLG,@C01:8,1,8) |
| s.eq | (rslt,arg1,arg2:0,8,0,16,0,16: ) | | |

```
s.equivalenc(rslt,arg1,arg2:0,8,0,8,0,8: )          s.equivalenc    (@T01,@T01,@T02:8,8,8)
s.exitproc  (nam ::  )                              s.exitproc      (KEYINMAIN:)
s.fixedwait (time:0,1275: )                         s.fixedwait     (10)
s.forcons   (indx,lwr,upr,slab,elab,val:0,8,0,8,0,8:) s.forcons     (COUNT,@C01,@C05,@11,@12:8,8,8,4)
s.forend    (indx,slab,elab:0,8:  )                 s.forend        (COUNT,@11,@12:8,4)
s.ge        (rslt,arg1,arg2:0,8,0,8,0,8:  )         s.ge            (@T01,AC3,@C02:8,8,8)
s.ge        (rslt,arg1,arg2:0,8,0,16,0,16:  )
s.gt        (rslt,arg1,arg2:0,8,0,8,0,8:  )         s.gt            (@T01,AC4,@C02:8,8,8)
s.gt        (rslt,arg1,arg2:0,8,0,16,0,16:  )
s.implicate (rslt,arg1,arg2:0,8,0,8,0,8: )          s.implicate     (@T01,@T01,@T02:8,8,8)
s.in        (:: )**                                 s.in            (1800000)
                                                    s.in/outport    (MSGVDT,TTL:8)
s.inputport (innam,tech:0,8: )                      s.inputport     (KEYFLG,TTL:8)
s.issuevent (outnm:0,8: )                           s.issuevent     (MENU:8)
s.jmpf      (val,loc :0,8:  )                       s.jmpf          (@T01,@01:8)
s.le        (rslt,arg1,arg2:0,8,0,8,0,8:  )         s.le            (@T01,AC2,@C02:8,8,8)
s.le        (rslt,arg1,arg2:0,8,0,16,0,16:  )
s.loc       (loc ::  )                              s.loc           (@01:)
s.lt        (rslt,arg1,arg2:0,8,0,8,0,8:  )         s.lt            (@T01,AC1,@C02:8,8,8)
s.lt        (rslt,arg1,arg2:0,8,0,16,0,16:  )
s.main      ( ::  )*
s.monitor   ( ::  )*
s.mult      (rslt,arg1,arg2:0,8,0,8,0,8:  )         s.mult          (@T01,ACNUM,@C07:8,8,8)
s.mult      (rslt,arg1,arg2:0,16,0,8,0,8:  )
s.mult      (rslt,arg1,arg2:0,16,0,16,0,16:  )
s.ne        (rslt,arg1,arg2:0,8,0,8,0,8:  )         s.ne            (@T01,AC0,@C02:8,8,8)
```

| | | | |
|---|---|---|---|
| s.ne | (rslt,arg1,arg2:0,8,0,16,0,16: ) | | |
| s.ni | (:: )** | s.ni | (::) |
| s.not | (rslt,arg1:0,8,0,8: ) | s.not | ???? |
| s.or | (rslt,arg1,arg2:0,8,0,8,0,8: ) | s.or | (@T01,@T01,@T02:8,8,8) |
| s.outputport | (outnm,tech:0,8: ) | s.outputport | (MENU,TTL:8) |
| s.proc | (nam :: ) | s.proc | (KEYINMAIN:) |
| s.sensesecond | (innam:0,8: ) | s.sensesecond | (KEYFLG:1) |
| s.setime | (clktim:0,32768: )*** | | |
| s.stboolwait | (top,maxtm:: ) | s.stboolwait | (@09:) |
| s.sub | (rslt,arg1,arg2:0,8,0,8,0,8: ) | s.sub | (@T01,@C01,@C02:8,8,8) |
| s.sub | (rslt,arg1,arg2:0,16,0,16,0,16: ) | | |
| s.tabaccp2 | (:: )* | | |
| s.tabend | ( :: )* | | |
| s.tabent | (fnc,task :: )* | | |
| s.var | (name:0,8: ) | s.var | (KEYINMAIN:8,0)**** |
| s.var | (name:0,16: ) | s.var | (@T01:8)**** |
| s.waitleast | (indx,upr,top,bot,per,max:0,8,0,8: ) | s.waitleast | (@T01,8:500) |
| s.whend | (top,bot:: ) | s.whend | (@17,@18:) |
| s.whilecon | (rslt,bot:0,8: ) | s.whilecon | (@T01,@18:4) |
| s.whilestart | (top,lpct:: ) | s.whilestart | (@17:) |

* These primitives are used by the CSDE system to construct the monitor section of the generated controller program. Although they must be present in the Realization Volume, they are not produced by the Translator because the monitor strategy is not controlled by the designer who writes the CSDL problem.

** These primitives were added to the Realization Volume for completeness but are not useable as currently implemented.

*** This primitive is called by 's.atboolwait'.

**** The Translator produces 2 versions of 's.var' and both are compatible with the Realization Volume.

???? This Translator primitive was not generated by Carson's original CSDL test program and subseqent attempts to generate it proved unsuccessful.

# APPENDIX E

## REVISED Z-80 REALIZATION VOLUME

This appendix displays the revised Z-80 Realization Volume. It contains primitives retained from Smith's original Z-80 Volume plus newly constructed primitives.

```
v0000 z80 cpu          : clpper=0.25 : memdly=0.25 : moncst=10:
v0872h.cardcage  (:: , , ,0,0,872,876)
v1005h.clock     (:: , , ,0,0,1005,1028)
v1029h.keydisplay(:: , , ,6,8,1029,1048)
v0877h.memory    (:: , , ,2,3,877,897)
v0847h.processor (:: , , ,2,3,847,871)
v0898h.tcardcage (:: , , ,0,0,898,904)
v1228h.uart      (:: , , ,0,0,1228,1242)
v1206s. call     (nam ::3,17,5,7,0,1206,1213)
v0354s.add       (rslt,arg1,arg2:0,8,0,8,0,8:23,78,26,14,0,354,368)
v0553s.add       (rslt,arg1,arg2:0,16,0,16,0,16:31,126,37,18,0,553,571)
v0110s.and       (rslt,arg1,arg2:0,8,0,8,0,8:11,47,14,10,0,110,120)
v0572s.assign    (var,data:0,8,0,8:6,26,8,7,0,572,579)
v0580s.assign    (var,data:0,16,0,16:6,32,10,7,0,580,587)
v1182s.boolwait  (rslt,top,bot:0,8:22,93,26,23,0,1182,1205)
v0596s.cons      (nam,val, :0,8:1,0,0,6,0,596,602)
v0677s.cons      (nam,val, :0,16:2,0,0,6,0,677,683)
v0905s.divide    (rslt,arg1,arg2:0,8,0,8,0,8:56,504,129,41,0,905,946)
v0947s.divide    (rslt,arg1,arg2:0,16,0,16,0,16:80,1465,376,57,0,947,1004)
v0700s.end       (::3,10,3,8,10,700,710)
v0228s.eq        (rslt,arg1,arg2:0,8,0,8,0,8:16,70,20,13,0,228,241)
v0441s.eq        (rslt,arg1,arg2:0,8,0,16,0,16:18,91,26,13,0,441,454)
v0148s.equivalenc(rslt,arg1,arg2:0,8,0,8,0,8:12,51,15,17,0,148,165)
v0670s.exitproc  (nam ::1,10,3,6,0,670,676)
v1049s.fixedwait (time:0,1275:15,-5,18,6,0,1049,1068)
v0396s.forcons   (indx,lwr,upr,slab,elab,val:0,8,0,8,0,8:17,70,21,6,0,396,413)
v0315s.forend    (indx,slab,elab:0,8:7,27,8,3,0,315,324)
v0286s.ge        (rslt,arg1,arg2:0,8,0,8,0,8:42,108,31,28,0,286,314)
v0414s.ge        (rslt,arg1,arg2:0,8,0,16,0,16:46,118,34,26,0,414,440)
v0080s.gt        (rslt,arg1,arg2:0,8,0,8,0,8:45,118,34,29,0,80,109)
v0482s.gt        (rslt,arg1,arg2:0,8,0,16,0,16:46,118,34,26,0,482,508)
v0166s.implicate (rslt,arg1,arg2:0,8,0,8,0,8:14,57,17,16,0,166,182)
v1214s.in        (:: , , ,9,0,1214,1223)
v1104s.inputport (innam,tech:0,8:0,0,0,13,12,1104,1125)
v1069s.issuevent (outnm:0,8:5,24,7,8,0,1069,1077)
v0711s.jmpf      (val,loc :0,8: 8,30,8,8,0,711,719)
v0325s.le        (rslt,arg1,arg2:0,8,0,8,0,8:42,108,31,28,0,325,353)
v0455s.le        (rslt,arg1,arg2:0,8,0,16,0,16:46,118,34,26,0,455,481)
v0693s.loc       (loc ::1,4,1,6,0,693,699)
v0242s.lt        (rslt,arg1,arg2:0,8,0,8,0,8:45,118,34,29,0,242,271)
```

```
v0369s.lt          (rslt,arg1,arg2:0,8,0,16,0,16:46,131,38,26,0,369,395)
v0509s.main        (::7,24,7,21,23,509,552)
v0720s.monitor     (::1,4,1,7,0,720,727)
v0735s.mult        (rslt,arg1,arg2:0,8,0,8,0,8:35,528,138,22,0,735,757)
v0758s.mult        (rslt,arg1,arg2:0,16,0,8,0,8:34,527,138,21,0,758,779)
v0780s.mult        (rslt,arg1,arg2:0,16,0,16,0,16:39,1105,289,22,0,780,802)
v0272s.ne          (rslt,arg1,arg2:0,8,0,8,0,8:16,71,20,13,0,272,285)
v0833s.ne          (rslt,arg1,arg2:0,8,0,16,0,16:18,91,26,13,0,833,846)
v1224s.ni          (:: , , ,3,0,1224,1227)
v0219s.not         (rslt,arg1:0,8,0,8:7,30,9,8,0,219,227)
v0069s.or          (rslt,arg1,arg2:0,8,0,8,0,8:11,47,14,10,0,69,79)
v1078s.outputport (outnm,tech:0,8:6,29,8,14,13,1078,1103)
v0588s.proc        (nam ::1,4,1,7,0,588,595)
v1126s.sensecond  (innam:0,8:56,129,37,44,0,1126,1170)
v0183s.setime      (clktim:0,32768:37,166,46,12,13,183,218)
v1171s.stboolwait (top,maxtm::1,-5,1,10,6,1171,1181)
v0635s.sub         (rslt,arg1,arg2:0,8,0,8,0,8:23,87,26,14,0,635,649)
v0650s.sub         (rslt,arg1,arg2:0,16,0,16,0,16:31,126,37,19,0,650,669)
v0614s.tabaccp2   (:: , , ,0,0,614,625)
v0728s.tabend      ( :: 3,10,3,6,0,728,734)
v0603s.tabent      (fnc,task ::10,51,15,10,0,603,613)
v0626s.var         (name:0,8:0,0,0,3,0,626,634)
v0684s.var         (name:0,16:0,0,0,3,0,684,692)
v0121s.waitleast  (indx,upr,top,bot,per,max:0,8,0,8:23,-10,27,26,20,121,147)
v0824s.whend       (top,bot::3,10,3,4,0,824,832)
v0815s.whilecon   (rslt,bot:0,8:7,27,8,8,0,815,823)
v0803s.whilestart(top,lpct::1,4,1,6,0,803,814)
v0068 .end index
v0069s.or          (rslt,arg1,arg2:0,8,0,8,0,8:11,47,14,10,0,69,79)
v0070com primitive to perform logical or
v0071com list=result,argument 1, argument 2 ::stor,time,ext,c,i,addrs
v0072begin stext
v0073ld  a,((arg1)) ;4m    13t    3b     rslt = arg1 .or. arg2
v0074ld  b,  a      ;1m    4t     1b
v0075ld  a,((arg2)) ;4m    13t    3b
v0076or  b          ;1m    4t     1b
v0077ld  ((rslt)),a ;4m    13t    3b
v0078endtext
v0079calc romptr=romptr+11
v0080s.gt          (rslt,arg1,arg2:0,8,0,8,0,8:45,118,34,29,0,80,109)
v0081com primitive to perform comparision between 2  8-bit numbers
v0082com list=result,argument 1, argument 2 ::stor,time,ext,c,i,addrs
v0083begin stext
v0084ld  a,((arg2)) ;4m    13t    3b     if arg2 lt arg1 then rslt=ffh
v0085ld  b,  a      ;1m    4t     1b     b=arg2
v0086ld  a,((arg1)) ;4m    13t    3b
v0087ld  c,  a      ;1m    4t     1b     c=arg1
v0088and a          ;1m    4t     1b     set sign flag of arg1
v0089jp  p,$+00dh   ;3m    10t    3b     jump if arg1 is positive
v0090ld  a,  b      ;1m    4t     1b     arg1 = -
v0091and a          ;1m    4t     1b     set sign flag of arg2
```

```
v0092ld  b,  c       ;1m    4t    1b    arg2 .swap. arg1
v0093jp  m,  $+011h  ;3m    10t   3b    arg2 = - arg1 = - comp backwards
v0094ld  a,  0       ;2m    7t    2b    arg2 = + arg1 = - false
v0095jr  $+22        ;3m    12t   2b
v0096ld  a,  b       ;1m    4t    1b
v0097and a           ;1m    4t    1b    set sign flag of arg2
v0098ld  a,  c       ;1m    4t    1b    restore arg1 to accumulator
v0099jp  p,  $+007h  ;3m    10t   3b    arg2 = + arg1 = +
v0100ld  a,11111111b;2m    7t    2b    arg2 = - arg1 = + true
v0101jr  $+12        ;3m    12t   2b
v0102cp  b           ;1m    4t    1b
v0103ld  a,00000000b;2m    7t    2b    result false arg2 )= arg1
v0104jp  z,  $+7     ;3m    10t   3b
v0105jp  m,  $+4     ;3m    10t   3b
v0106cpl             ;1m    4t    1b    result true arg2 lt  arg1
v0107ld ((rslt)),a   ;4m    13t   3b
v0108endtext
v0109calc romptr=romptr+45
v0110s.and       (rslt,arg1,arg2:0,8,0,8,0,8:11,47,14,10,0,110,120)
v0111com primitive to perform logical and
v0112com list=result,argument 1, argument 2 ::stor,time,ext,c,i,addrs
v0113begin stext
v0114ld  a,((arg1)) ;4m    13t   3b    rslt = arg1 .and. arg2
v0115ld  b,  a       ;1m    4t    1b
v0116ld  a,((arg2)) ;4m    13t   3b
v0117and b           ;1m    4t    1b
v0118ld  ((rslt)),a ;4m    13t   3b
v0119endtext
v0120calc romptr=romptr+11
v0121s.waitleast (indx,upr,top,bot,per,max:0,8,0,8:23,-10,27,26,20,121,147)
v0122com primitive to generate a software wait based on the results of
v0123com of an aritmetic expression whose integer result is passed to
v0124com waitleast in  upr     the value in upr is the number of times
v0125com fixedwait will be executed similar to a for loop     fixedwait
v0126com will be fed the time in the variable   per       top and bot
v0127com are labels     max   is the max time allowed specified by the
v0128com designer for all possible combinations of upr and per
v0129com per and max are in ms to the nearest 5ms  the max allowed value
v0130com for per is 1275ms and the max allowed value of upr is 127
v0131attr time=(max)*4000
v0132begin stext
v0133ld a,1              ;2m  7t  2b  counter always starts at one
v0134(top):ld ((indx)),a ;4m 13t  3b  update (indx) with latest value
v0135ld a,((upr))        ;4m 13t  3b
v0136ld b,a              ;1m  4t  1b
v0137ld a,((indx))       ;4m 13t  3b
v0138cp b                ;1m  4t  1b  compare to upper limit
v0139jp z,(bot)+3        ;3m 10t  3b  jump out of loop on indx=upr
v0140endtext
v0141call s.fixedwait ((per):)
v0142begin stext
```

```
v01431ld a,((indx))       ;4m 13t  3b  get current indx value
v0144inc a                ;1m  4t  1b  crank indx
v0145(bot):jp (top)       ;3m 10t  3b  jumpt to top of loop
v0146endtext
v0147calc romptr=romptr+23
v0148s.equivalenc(rslt,arg1,arg2:0,8,0,8,0,8:12,51,15,17,0,148,165)
v0149com primitive perform to the logical equivalence relation
v0150com the truth table is as follows
v0151com          arg1            arg2          rslt
v0152com          false(00h)      false         true(ffh)
v0153com          false           true          false
v0154com          true            false         false
v0155com          true            true          true
v0156com equivalence is simply the opposite of xor
v0157begin stext
v0158ld a,((arg1))     ;4m   13t   3b    rslt = arg1 .equiv. arg2
v0159ld b, a           ;1m    4t   1b
v0160ld a,((arg2))     ;4m   13t   3b
v0161xor b             ;1m    4t   1b
v0162cpl               ;1m    4t   1b
v0163ld ((rslt)),a     ;4m   13t   3b
v0164endtext
v0165calc romptr=romptr+12
v0166s.implicate (rslt,arg1,arg2:0,8,0,8,0,8:14,57,17,16,0,166,182)
v0167com primitive to perform logical implication check
v0168com truth table is as follows
v0169com          arg1              arg2           rslt
v0170com         false(00h)        false          true(ffh)
v0171com         false             true           true
v0172com         true              false          false
v0173com         true              true           true
v0174begin stext
v0175ld a,((arg2))     ;4m   13t   3b    rslt = arg1 .implicate. arg2
v0176and a             ;1m    4t   1b   set zero flag
v0177jp nz,$+7         ;3m   10t   3b  if arg2=true then rslt=true
v0178ld a,((arg1))     ;4m   13t   3b  if arg2=false then get arg1 and cpl it
v0179cpl               ;1m    4t   1b  rslt= .not. arg1
v0180ld ((rslt)),a     ;4m   13t   3b
v0181endtext
v0182calc romptr=romptr+14
v0183s.setime     (clktim:0,32768:37,166,46,12,13,183,218)
v0184com primitive to set channel 1 of ctc to some initial value
v0185com clktim is initial time decimal in milliseconds
v0186com because channel 0 serves as the clock input to channel 1
v0187com with 1 millisecond pulses there would be a latent delay in
v0188com resetting channel 1 because new values for the downcounter
v0189com are not transfered from the load register to the downcounter
v0190com until a new clock pulse is sensed    therefore this primitive
v0191com also short times the channel 0 clock to generate an output pulse such
v0192com that channel 1 is immediately reset to the value passed  through the
v0193com argument  clktim
```

```
v0194if clock .ne. 0 skip 2
v0195calc clock=1
v0196incl h.clock      (::)
v0197begin stext
v0198ld a,01110001b    ;2m   7t   2b  counter1+load lsb then msb+mode0+hex
v0199out (0f3h),a      ;3m  11t   2b  set mode control
v0200ld hl,(clktim)    ;3m  10t   3b  get time period
v0201ld a,l            ;1m   4t   1b  lsb of clktim
v0202out (0f1h),a      ;3m  11t   2b  load lsb to ctc channel 1
v0203ld a,h            ;1m   4t   1b  msb of clktim
v0204out (0f1h),a      ;3m  11t   2b  load msb to ctc channel 1
v0205ld a,00110100b    ;2m   7t   2b  countr0+load lsb then msb+mode2+bcd
v0206out (0f3h),a      ;3m   11t  2b  set mode control
v0207ld a,02h          ;2m   7t   2b  lsb of 0002 bcd
v0208out (0f0h),a      ;3m  11t   2b  02h in load reg lsb
v0209ld a,00h          ;2m   7t   2b  msb of 0002 bcd
v0210out (0f0h),a      ;3m  11t   2b  00h in load reg msb
v0211ld a,00110100b    ;2m   7t   2b  contr0+load lsb then msb+mode2+bcd
v0212out (0f3h),a      ;3m  11t   2b  set mode control
v0213ld a,00h          ;2m   7t   2b  lsb of 2000 bcd
v0214out (0f0h),a      ;3m  11t   2b  00h in load reg lsb
v0215ld a,20h          ;2m   7t   2b  msb of 2000 bcd
v0216out (0f0h),a      ;3m  11t   2b  20h in load reg msb
v0217endtext
v0218calc romptr=romptr+37
v0219s.not       (rslt,arg1:0,8,0,8:7,30,9,8,0,219,227)
v0220com primitive to perform logical not, complement
v0221com list=result,argument 1, argument 2 ::stor,time,ext,c,i,addrs
v0222begin stext
v0223ld  a,((arg1)) ;4m    13t    3b     rslt = not arg1
v0224cpl            ;1m    4t     1b
v0225ld  ((rslt)),a ;4m    13t    3b
v0226endtext
v0227calc romptr=romptr+7
v0228s.eq        (rslt,arg1,arg2:0,8,0,8,0,8:16,70,20,13,0,228,241)
v0229com primitive to perform comparision between 2  8-bit numbers
v0230com list=result,argument 1, argument 2 ::stor,time,ext,c,i,addrs
v0231begin stext
v0232ld  a,  ((arg1)) ;4m    13t    3b     if arg1 = arg2 then rslt=ffh
v0233ld  b,  a        ;1m    4t     1b
v0234ld  a,  ((arg2)) ;4m    13t    3b
v0235cp  b            ;1m    4t     1b
v0236ld  a,11111111b;2m    7t     2b
v0237jr  z, $+3       ;3m    12t    2b     result equal
v0238cpl              ;1m    4t     1b     result not equal
v0239ld ((rslt)),a  ;4m    13t    3b
v0240endtext
v0241calc romptr=romptr+16
v0242s.lt        (rslt,arg1,arg2:0,8,0,8,0,8:45,118,34,29,0,242,271)
v0243com primitive to perform comparision between 2  8-bit numbers
v0244com list=result,argument 1, argument 2 ::stor,time,ext,c,i,addrs
```

```
v0245begin stext
v0246ld  a,((arg1)) ;4m    13t    3b     if arg1 lt arg2 then rslt=ffh
v0247ld  b,  a      ;1m    4t     1b     b=arg1
v0248ld  a,((arg2)) ;4m    13t    3b
v0249ld  c,  a      ;1m    4t     1b     c=arg2
v0250and a          ;1m    4t     1b     set sign flag of arg2
v0251jp  p,$+00dh   ;3m    10t    3b     jump if arg2 is positive
v0252ld  a,  b      ;1m    4t     1b     arg2 = -
v0253and a          ;1m    4t     1b     set sign flag of arg1
v0254ld  b,  c      ;1m    4t     1b     arg1 .swap. arg2
v0255jp  m,$+011h   ;3m    10t    3b     arg1 = - arg2 = - comp backwards
v0256ld  a,  0      ;2m    7t     2b     arg1 = + arg2 = - false
v0257jr  $+016h     ;3m    12t    2b
v0258ld  a,  b      ;1m    4t     1b
v0259and a          ;1m    4t     1b     set sign flag of arg1
v0260ld  a,  c      ;1m    4t     1b     restore arg2 to accumulator
v0261jp  p,$+007h   ;3m    10t    3b     arg1 = + arg2 = +
v0262ld  a,11111111b;2m    7t     2b     arg1 = - arg2 = + true
v0263jr  $+00ch     ;3m    12t    2b
v0264cp  b          ;1m    4t     1b
v0265ld  a,00000000b;2m    7t     2b     result false arg1 )= arg2
v0266jp  z,  $+7    ;3m    10t    3b
v0267jp  m,  $+4    ;3m    10t    3b
v0268cpl            ;1m    4t     1b     result true arg1 lt  arg2
v0269ld ((rslt)),a  ;4m    13t    3b
v0270endtext
v0271calc romptr=romptr+45
v0272s.ne        (rslt,arg1,arg2:0,8,0,8,0,8:16,71,20,13,0,272,285)
v0273com primitive to perform comparision between 2  8-bit numbers
v0274com list=result,argument 1, argument 2 ::stor,time,ext,c,i,addrs
v0275begin stext
v0276ld  a,((arg1)) ;4m    13t    3b     if arg1 = arg2 then rslt=ffh
v0277ld  b,  a      ;1m    4t     1b
v0278ld  a,((arg2)) ;4m    13t    3b
v0279cp  b          ;1m    4t     1b
v0280ld  a,  0      ;2m    7t     2b
v0281jr  z,  $+003h ;3m    13t    2b     result not equal
v0282cpl            ;1m    4t     1b     result equal
v0283ld  ((rslt)),a ;4m    13t    3b
v0284endtext
v0285calc romptr=romptr+16
v0286s.ge        (rslt,arg1,arg2:0,8,0,8,0,8:42,108,31,28,0,286,314)
v0287com primitive to perform comparision between 2  8-bit numbers
v0288com list=result,argument 1, argument 2 ::stor,time,ext,c,i,addrs
v0289begin stext
v0290ld  a,((arg2)) ;4m    13t    3b     if arg2 le arg1 then rslt=ffh
v0291ld  b,  a      ;1m    4t     1b     b=arg2
v0292ld  a,((arg1)) ;4m    13t    3b
v0293ld  c,  a      ;1m    4t     1b     c=arg1
v0294and a          ;1m    4t     1b     set sign flag of arg1
v0295jp  p,$+00dh   ;3m    10t    3b     jump if arg1 is positive
```

```
v0296ld  a,  b       ;1m    4t     1b     arg1 = -
v0297and a           ;1m    4t     1b     set sign flag of arg2
v0298ld  b,  c       ;1m    4t     1b     arg2 .swap. arg1
v0299jp  m,  $+011h  ;3m    10t    3b     arg2 = - arg1 = - comp backwards
v0300ld  a,  0       ;2m    7t     2b     arg2 = + arg1 = - false
v0301jr  $+013h      ;3m    12t    2b
v0302ld  a,  b       ;1m    4t     1b
v0303and a           ;1m    4t     1b     set sign flag of arg2
v0304ld  a,  c       ;1m    4t     1b     restore arg1 to accumulator
v0305jp  p,  $+007h  ;3m    10t    3b     arg2 = + arg1 = +
v0306ld  a,11111111b;2m    7t     2b     arg2 = - arg1 = + true
v0307jr  $+009h      ;3m    12t    2b
v0308cp  b           ;1m    4t     1b
v0309ld  a,11111111b;2m    7t     2b     result false arg2 )= arg1
v0310jp  p,  $+4     ;3m    10t    3b
v0311cpl             ;1m    4t     1b     result true arg2 lt  arg1
v0312ld ((rslt)),a   ;4m    13t    3b
v0313endtext
v0314calc romptr=romptr+42
v0315s.forend     (indx,slab,elab:0,8:7,27,8,3,0,315,324)
v0316com primitive to end a for loop
v0317com list=index,start label,end label
v0318calc pop reps
v0319begin stext
v0320ld  a,((indx)) ;4m    13t    3b     get value of index at top of loop
v0321inc a          ;1m    4t     1b     crank index
v0322(elab):jp (slab) ;3m    10t    3b     jump to for loop test
v0323endtext
v0324calc romptr=romptr+7
v0325s.le         (rslt,arg1,arg2:0,8,0,8,0,8:42,108,31,28,0,325,353)
v0326com primitive to perform comparision between 2  8-bit numbers
v0327com list=result,argument 1, argument 2 ::stor,time,ext,c,i,addrs
v0328begin stext
v0329ld  a,((arg1)) ;4m    13t    3b     if arg1 le arg2 then rslt=ffh
v0330ld  b,  a      ;1m    4t     1b     b=arg1
v0331ld  a,((arg2)) ;4m    13t    3b
v0332ld  c,  a      ;1m    4t     1b     c=arg2
v0333and a          ;1m    4t     1b     set sign flag of arg2
v0334jp  p,$+13     ;3m    10t    3b     jump if arg2 is positive
v0335ld  a,  b      ;1m    4t     1b     arg2 = -
v0336and a          ;1m    4t     1b     set sign flag of arg1
v0337ld  a,  c      ;1m    4t     1b     restore arg2 to accumulator
v0338jp  m,  $+17   ;3m    10t    3b     arg1 = - arg2 = - comp backwards
v0339ld  a,  0      ;2m    7t     2b     arg1 = + arg2 = - false
v0340jr  $+13       ;3m    12t    2b
v0341ld  a,  b      ;1m    4t     1b
v0342and a          ;1m    4t     1b     set sign flag of arg1
v0343ld  a,  c      ;1m    4t     1b     restore arg2 to accumulator
v0344jp  p,  $+7    ;3m    10t    3b     arg1 = + arg2 = +
v0345ld  a,11111111b;2m    7t     2b     arg1 = - arg2 = + true
v0346jr  $+9        ;3m    12t    2b
```

```
v0347cp  b          ;1m    4t     1b
v03481d  a,11111111b;2m    7t     2b     result false arg1 )= arg2
v0349jp  p,  $+4    ;3m    10t    3b
v0350cpl            ;1m    4t     1b     result true arg1 lt  arg2
v0351ld  ((rslt)),a ;4m    13t    3b
v0352endtext
v0353calc romptr=romptr+42
v0354s.add       (rslt,arg1,arg2:0,8,0,8,0,8:23,78,26,14,0,354,368)
v0355com primitive to add arg1 and arg2 and store in rslt
v0356com list=rslt,arg1,arg2:precisions:s,t,e,c,i,addr
v0357begin stext
v0358ld a, ((arg1)) ;13t   4m 3b  store arg1 in accumulator
v0359ld hl,(arg2)   ;10t   3m 3b  have hl point to arg2 byte
v0360add a, (hl)    ;7t    2m 1b  add accumulator with arg2
v0361jp po ,$+13    ;3m   10t 3b  if no overflow store result
v0362jp c  ,$+8     ;3m   10t 3b  if carry the maximize minus rslt
v03631d  a,01111111b;2m    7t 2b  put in largest positive value
v0364jp $+5         ;3m   10t 3b
v03651d a,10000000b ;2m    7t 2b  put in largest negative value
v0366ld ((rslt)),a  ;13t   4m 3b  save result of add in rslt
v0367endtext
v0368calc romptr=romptr+23
v0369s.lt        (rslt,arg1,arg2:0,8,0,16,0,16:46,131,38,26,0,369,395)
v0370com primitive to perform comparision between 2 16-bit numbers
v0371com list=result,argument 1, argument 2 ::stor,time,ext,c,i,addrs
v0372begin stext
v0373ld de,((arg1)) ;6m 20t 4b if arg1 lt arg2 then rslt=ffh  de=(arg1)
v0374ld hl,((arg2)) ;5m 16t 3b                               hl=(arg2)
v0375ld  a,  h      ;1m  4t 1b
v0376and a          ;1m 4t  1b     set sign flag of arg2
v0377jp  p,$+13     ;3m 10t 3b     jump if arg2 is positive
v0378ld  a,  d      ;1m 4t  1b     arg2 = -
v0379and a          ;1m 4t  1b     set sign flag of arg1
v0380jp  m,$+18     ;3m 10t 3b     arg1 = - arg2 = - comp backwards
v0381ld  a,  0      ;2m 7t  2b     arg1 = + arg2 = - false
v0382jp  $+24       ;3m 10t 3b
v0383ld  a,  d      ;1m  4t 1b
v0384and a          ;1m 4t  1b     set sign flag of arg1
v0385jp  p,$+8      ;3m 10t 3b     arg1 = + arg2 = +
v0386ld  a,11111111b;2m 7t  2b     arg1 = - arg2 = + true
v0387jp  $+14       ;3m 10t 3b
v0388sbc hl,de      ;4m 15t 2b
v03891d  a,00000000b;2m 7t  2b     result false arg1 )= arg2
v0390jp  z,  $+7    ;3m 10t 3b
v0391jp  m,  $+4    ;3m 10t 3b
v0392cpl            ;1m 4t  1b     result true arg1 lt  arg2
v0393ld ((rslt)),a  ;4m 13t 3b
v0394endtext
v0395calc romptr=romptr+46
v0396s.forcons   (indx,lwr,upr,slab,elab,val:0,8,0,8,0,8:17,70,21,6,0,396,413)
v0397com primitive to set up a loop with constant bounds
```

```
v0398com list=index,lower bound,upper bound,start label,end label
v0399com     max allowed value of indx,lwr,and,upr is 127
v0400com    because the translator calls for 16 bit precision if a
v0401com    greater number is specified   (val) is max loop count
v0402calc push reps
v0403calc reps=(val)
v0404begin stext
v0405ld  a, ((lwr))       ;4m 13t 3b     lower bound of counter
v0406(slab):ld ((indx)),a ;4m 13t 3b     update (indx) with latest value
v0407ld a,((upr))         ;4m 13t 3b
v0408ld b,a               ;1m  4t 1b
v0409ld a,((indx))        ;4m 13t 3b
v0410cp b                 ;1m  4t 1b     compare to upper limit
v0411jp z,(elab)+3        ;3m 10t 3b     jump out of loop on index=upr
v0412endtext
v0413calc romptr=romptr+17
v0414s.ge        (rslt,arg1,arg2:0,8,0,16,0,16:46,118,34,26,0,414,440)
v0415com primitive to perform comparision between 2 16-bit numbers
v0416com list=result,argument 1, argument 2 ::stor,time,ext,c,i,addrs
v0417begin stext
v0418ld de,((arg2)) ;6m 20t 4b if arg2 learg1 then rslt=ffh  de=(arg2)
v0419ld hl,((arg1)) ;5m 16t 3b                               hl=(arg1)
v0420ld  a,  h      ;1m  4t 1b
v0421and a          ;1m    4t     1b     set sign flag of arg1
v0422jp  p,$+13     ;3m    10t    3b     jump if arg1 is positive
v0423ld  a,  d      ;1m    4t     1b     arg1 = -
v0424and a          ;1m    4t     1b     set sign flag of arg2
v0425jp  m,$+18     ;3m    10t    3b     arg2 = - arg1 = - comp backwards
v0426ld  a,  0      ;2m    7t     2b     arg2 = + arg1 = - false
v0427jp  $+24       ;3m 10t 3b
v0428ld  a,  d      ;1m  4t 1b
v0429and a          ;1m    4t     1b     set sign flag of arg2
v0430jp  p,$+8      ;3m    10t    3b     arg2 = + arg1 = +
v0431ld  a,11111111b;2m    7t     2b     arg2 = - arg1 = + true
v0432jp  $+14       ;3m 10t 3b
v0433sbc hl,de      ;4m 15t 2b
v0434ld  a,00000000b;2m    7t     2b     result false arg2 )= arg1
v0435jp  m,  $+7    ;3m    10t    3b
v0436jp  m,  $+4    ;3m    10t    3b
v0437cpl            ;1m    4t     1b     result true arg2 lt  arg1
v0438ld ((rslt)),a  ;4m    13t    3b
v0439endtext
v0440calc romptr=romptr+46
v0441s.eq        (rslt,arg1,arg2:0,8,0,16,0,16:18,91,26,13,0,441,454)
v0442com primitive to perform comparision between 2 16-bit numbers
v0443com list=result,argument 1, argument 2 ::stor,time,ext,c,i,addrs
v0444begin stext
v0445ld de,((arg1)) ;6m 20t 4b if arg1 = arg2 then rslt=ffh  de=(arg1)
v0446ld hl,((arg2)) ;5m 16t 3b                               hl=(arg2)
v0447and a          ;1m  4t 1b  clear carry flag
v0448sbc hl,de      ;4m 15t 2b
```

```
v0449ld  a,11111111b;2m  7t 2b
v0450jr  z, $+3     ;3m 12b 2b     result equal
v0451cpl            ;1m  4t 1b     result not equal
v0452ld ((rslt)),a  ;4m 13t 3b
v0453endtext
v0454calc romptr=romptr+18
v0455s.le        (rslt,arg1,arg2:0,8,0,16,0,16:46,118,34,26,0,455,481)
v0456com primitive to perform comparision between 2 16-bit numbers
v0457com list=result,argument 1, argument 2 ::stor,time,ext,c,i,addrs
v0458begin stext
v0459ld de,((arg1)) ;6m 20t 4b if arg1 learg2 then rslt=ffh  de=(arg1)
v0460ld hl,((arg2)) ;5m 16t 3b                               hl=(arg2)
v0461ld  a,  h      ;1m  4t 1b
v0462and a          ;1m    4t     1b     set sign flag of arg2
v0463jp  p,$+13     ;3m    10t    3b     jump if arg2 is positive
v0464ld  a,  d      ;1m    4t     1b     arg2 = -
v0465and a          ;1m    4t     1b     set sign flag of arg1
v0466jp  m,$+18     ;3m    10t    3b     arg1 = - arg2 = - comp backwards
v0467ld  a,  0      ;2m    7t     2b     arg1 = + arg2 = - false
v0468jp  $+24       ;3m 10t 3b
v0469ld  a,  d      ;1m  4t 1b
v0470and a          ;1m    4t     1b     set sign flag of arg1
v0471jp  p,$+8      ;3m    10t    3b     arg1 = + arg2 = +
v0472ld  a,11111111b;2m    7t     2b     arg1 = - arg2 = + true
v0473jp  $+14       ;3m 10t 3b
v0474sbc hl,de      ;4m 15t 2b
v0475ld  a,00000000b;2m    7t     2b     result false arg1 gt  arg2
v0476jp  m,  $+7    ;3m    10t    3b
v0477jp  m,  $+4    ;3m    10t    3b
v0478cpl            ;1m    4t     1b     result true arg1 le arg2
v0479ld ((rslt)),a  ;4m    13t    3b
v0480endtext
v0481calc romptr=romptr+46
v0482s.gt        (rslt,arg1,arg2:0,8,0,16,0,16:46,118,34,26,0,482,508)
v0483com primitive to perform comparision between 2 16-bit numbers
v0484com list=result,argument 1, argument 2 ::stor,time,ext,c,i,addrs
v0485begin stext
v0486ld de,((arg2)) ;6m 20t 4b if arg2 lt arg1 then rslt=ffh  de=(arg2)
v0487ld hl,((arg1)) ;5m 16t 3b                               hl=(arg1)
v0488ld  a,  h      ;1m  4t 1b
v0489and a          ;1m    4t     1b     set sign flag of arg1
v0490jp  p,$+13     ;3m    10t    3b     jump if arg1 is positive
v0491ld  a,  d      ;1m    4t     1b     arg1 = -
v0492and a          ;1m    4t     1b     set sign flag of arg2
v0493jp  m,$+18     ;3m    10t    3b     arg2 = - arg1 = - comp backwards
v0494ld  a,  0      ;2m    7t     2b     arg2 = + arg1 = - false
v0495jp  $+24       ;3m 10t 3b
v0496ld  a,  d      ;1m  4t 1b
v0497and a          ;1m    4t     1b     set sign flag of arg2
v0498jp  p,$+8      ;3m    10t    3b     arg2 = + arg1 = +
v0499ld  a,11111111b;2m    7t     2b     arg2 = - arg1 = + true
```

```
v0500jp  $+14       ;3m 10t 3b
v0501sbc hl,de      ;4m 15t 2b
v0502ld  a,00000000b;2m    7t     2b     result false arg2 gt  arg1
v0503jp  z,  $+7    ;3m    10t    3b
v0504jp  m,  $+4    ;3m    10t    3b
v0505cpl            ;1m    4t     1b     result true arg2 le arg1
v0506ld ((rslt)),a ;4m    13t    3b
v0507endtext
v0508calc romptr=romptr+46
v0509s.main      (::7,24,7,21,23,509,552)
v0510com primitive to define controller setup and initialization
v0511com list = empty : empty : storage, time, ext, calc, incl, addr
v0512com the rom pointer is set to start at 16384 or 4000h since this
v0513com is the beginning of user addressible memory in the pro-log
v0514com it is called rom because ultimately the controller's operating
v0515com program would be burned into rom   the ram pointer starts
v0516com at 32735 which is 32 bytes below the top of usable memory
v0517com on the pro-log  to allow a 32 byte stack  the top of user
v0518com addressable memory on the pro-log is 32767 or 7fffh
v0519com all initializations will be done through the use of global
v0520com variable initlk and linked labels
v0521com following the initializations program will jump to the top of the
v0522com polling loop for the task contingency pairs.
v0523com to allow the use of a debug prom developed at the naval
v0524com postgraduate school electrical engineering department the
v0525com starting location is changed to 4096 to allow a the system
v0526com to auto boot and to allow loading of memory from another
v0527com computer via the dual uart card.  the loading prom inhibits the
v0528com use of the reset location because of the location of the code and
v0529com the interrupt loactions used in a debugger for the prolog system
v0530calc romptr=16384
v0531calc ramptr=32735
v0532incl h.processor (::)
v0533incl h.cardcage  (::)
v0534begin stext
v0535;
v0536;             zilog z-80 based system
v0537;
v0538; #idsect#
v0539; #idsect#
v0540; #idsect#
v0541;
v0542;
v0543.z80
v0544aseg
v0545org (ramptr)          ;ram pointer is pointing to top of memory - stack
v0546@stak:defs 32              ;             32b    define stack area
v0547org   (romptr)        ;begin code after reserved interrupt area
v0548@cold:ld  sp,@stak+32      ;3m    10t    3b     initialize stack pointer
v0549di                    ;1m    4t     1b     disable maskable interrupts
v0550jp @i(initlk)         ;3m    10t    3b     do hardware initializations
```

```
v0551endtext
v0552calc romptr=romptr+7
v0553s.add      (rslt,arg1,arg2:0,16,0,16,0,16:31,126,37,18,0,553,571)
v0554com primitive to add arg1 and arg2 and store in rslt
v0555begin stext
v0556ld hl, ((arg1));5m    16t    3b     load arg1 in hl pair
v0557ld bc, ((arg2));6m    20t    4b     load arg2 in bc pair
v0558ld  a,  l      ;1m    4t     1b
v0559add a,  c      ;1m    4t     1b     add lsb
v0560ld  l,  a      ;1m    4t     1b
v0561ld  a,  h      ;1m    4t     1b
v0562adc a,  b      ;1m    4t     1b     add msb
v0563ld  h,  a      ;1m    4t     1b
v0564jp po ,$+15    ;3m   10t     3b  if no overflow store result
v0565jp c  ,$+9     ;3m   10t     3b  if carry the maximize minus rslt
v0566ld  hl, 7fffh  ;3m    10t    3b  put in largest positive value
v0567jp $+6         ;3m   10t     3b
v0568ld  hl, 8000h  ;3m    10t    3b  put in largest negative value
v0569ld  ((rslt)),hl;5m    16t    3b     save result
v0570endtext
v0571calc romptr=romptr+31
v0572s.assign    (var,data:0,8,0,8:6,26,8,7,0,572,579)
v0573com primitive to assign a value of one variable to another variable
v0574com list=var,data-var:var-prec,data-prec:stor,time,ext,calc,incl,addr
v0575begin stext
v0576ld  a,((data)) ;4m    13t    3b     assign (data)
v0577ld  ((var)),a  ;4m    13t    3b            to     (var)
v0578endtext
v0579calc    romptr =romptr + 6
v0580s.assign    (var,data:0,16,0,16:6,32,10,7,0,580,587)
v0581com primitive to assign a value of one variable to another variable
v0582com list=var,data-var:var-prec,data-prec:stor,time,ext,calc,incl,addr
v0583begin stext
v0584ld  hl,((data)) ;5m    16t    3b     assign (data)
v0585ld  ((var)),hl  ;5m    16t    3b            to     (var)
v0586endtext
v0587calc    romptr =romptr + 6
v0588s.proc      (nam ::1,4,1,7,0,588,595)
v0589com primitive to define procedure entry poinp
v0590com list=proc-name :empty:storage,time,ext,calc,incl,addr
v0591begin stext
v0592;procedure (nam)
v0593@(nam): nop    ;1m  4t     1b entry point for (nam)
v0594endtext
v0595calc romptr=romptr+1
v0596s.cons      (nam,val, :0,8:1,0,0,6,0,596,602)
v0597com primitive to define data
v0598com list=data-name,value:value-prec,stor,time,ext,c,i,addrs
v0599begin stext
v0600(nam):   defb  (val)  ;reserve one byte for data
v0601endtext
```

```
v0602calc romptr=romptr+1
v0603s.tabent     (fnc,task ::10,51,15,10,0,603,613)
v0604com primitive to add one entry to monitor table
v0605com list= func-name, task name:empty:s,t,e,c,i,address
v0606begin stext
v0607call  @(fnc)  ;5m    17t    3b     test for contingency (fnc)
v0608ld  a,((fnc))  ;4m    13t    3b     get contingency result
v0609cp  11111111b  ;1m    4t     1b     check if result true
v0610call z,@(task) ;5m    17t    3b     if true execute task
v0611               ;if not true get next tabent or tabend to loop
v0612endtext
v0613calc romptr=romptr+10
v0614s.tabaccp2  (:: , , ,0,0,614,625)
v0615com this is a dummy primitive to allow compatibility with the 8080
v0616com library.  the functions that would be performed in this primitive
v0617com are all located in s.tabent.  this has the effect of eliminating
v0618com intermediate table and increasing execution speed.  if there are
v0619com wide variations in contingency/task speeds more memory will be
v0620com than in the 8080 primitive.  note s.main is also changed because
v0621com of the elimination of the intermediate table
v0622com list= func-name, task name:empty:s,t,e,c,i,address
v0623begin stext
v0624    ; this space is deliberately void.  this is a dummy primitive.
v0625endtext
v0626s.var        (name:0,8:0,0,0,3,0,626,634)
v0627com primitive to define storage for 8 bit variable integer or logical
v0628com list=data-name,value:value-prec,stor,time,ext,c,i,addrs
v0629calc ramptr=ramptr - 1
v0630begin stext
v0631org  (ramptr)        ;8 bit variable (name) in ram
v0632(name):  defb  0     ;0m    0t     1b
v0633org  (romptr)
v0634endtext
v0635s.sub        (rslt,arg1,arg2:0,8,0,8,0,8:23,87,26,14,0,635,649)
v0636com primitive to subtract arg2 from arg1 and store in rslt
v0637com list= rslt,arg1,arg2:precisions:s,t,e,c,i,a
v0638begin stext
v0639ld a,((arg1))  ;4m    13t    3b     load arg1 in accumulator
v0640ld hl,(arg2)   ;3m    10t    3b     point hl to arg2
v0641sub (hl)       ;2m     7t    1b     arg1 - arg2
v0642jp po ,$+13    ;3m   10t 3b  if no overflow store result
v0643jp c  ,$+8     ;3m   10t 3b  if carry the maximize minus rslt
v0644ld  a,01111111b;2m    7t 2b  put in largest positive value
v0645jp $+5         ;3m   10t 3b
v0646ld a,10000000b ;2m    7t 2b  put in largest negative value
v0647ld ((rslt)),a  ;13t   4m 3b  save result of add in rslt
v0648endtext
v0649calc romptr=romptr+23
v0650s.sub        (rslt,arg1,arg2:0,16,0,16,0,16:31,126,37,19,0,650,669)
v0651com primitive to subract arg2 from arg1 and store answer in rslt
v0652com list=rslt,arg1,arg2:precisions:s,t,e,c,i,addr
```

```
v0653begin stext
v0654ld hl, ((arg1))  ;5m    16t    3b     load arg1 in hl pair
v0655ld bc, ((arg2))  ;6m    20t    4b     load arg2 in bc pair
v0656ld  a,  l        ;1m    4t     1b
v0657sub c            ;1m    4t     1b     subrtract lsb
v0658ld  l,  a        ;1m    4t     1b
v0659ld  a,  h        ;1m    4t     1b
v0660sbc a,  b        ;1m    4t     1b     subtract msb
v0661ld  h,  a        ;1m    4t     1b
v0662jp po ,$+15      ;3m    10t    3b  if no overflow store result
v0663jp c  ,$+9       ;3m    10t    3b  if carry the maximize minus rslt
v0664ld  hl, 7fffh    ;3m    10t    3b  put in largest positive value
v0665jp $+6           ;3m    10t    3b
v0666ld  hl, 8000h    ;3m    10t    3b  put in largest negative value
v0667ld  ((rslt)),hl  ;5m    16t    3b     save result
v0668endtext
v0669calc romptr=romptr+31
v0670s.exitproc  (nam ::1,10,3,6,0,670,676)
v0671com primitive to close proc
v0672com list=proc-nam, contnam:empty: storage, time, ext,calc,incl,addr
v0673begin stext
v0674ret              ;3m    10t    1b     return to monitor,exit (nam)
v0675endtext
v0676calc romptr=romptr+1
v0677s.cons      (nam,val, :0,16:2,0,0,6,0,677,683)
v0678com primitive to define data for 16 bit integer
v0679com list=data-name,value:value-prec,stor,time,ext,c,i,addrs
v0680begin stext
v0681(nam):   defw  (val)  ;define a two byte integer
v0682endtext
v0683calc romptr=romptr+2
v0684s.var       (name:0,16:0,0,0,3,0,684,692)
v0685com primitive to define storage for 16 bit variable integer
v0686com list=data-name,value:value-prec,stor,time,ext,c,i,addrs
v0687calc ramptr=ramptr - 2
v0688begin stext
v0689org  (ramptr)       ;16 bit variable (name) in ram
v0690(name): defw 0    ;0m    0t     2b
v0691org  (romptr)
v0692endtext
v0693s.loc       (loc ::1,4,1,6,0,693,699)
v0694com primitive to define a lable (location)
v0695com list=label-name :empty: storage,time,ext,calc,incl,addr
v0696begin stext
v0697(loc):  nop                ; define location (loc)
v0698endtext
v0699calc romptr=romptr+1
v0700s.end       (::3,10,3,8,10,700,710)
v0701com primitive to end software listing and complete implementation
v0702com list=empty:empty:stor,time,ext,calc,incl,addr
v0703begin stext
```

```
v0704@i(initlk):jp @spvsr ;3m 10t 3b  initialization of hardware is complete
v0705                     ;           start top of main monitor loop
v0706end                            ;end of software listing ready for assembly
v0707endtext
v0708calc romptr=romptr+3
v0709com put in memory needed for implementation in ram and rom
v0710incl h.memory    (::)
v0711s.jmpf      (val,loc :0,8: 8,30,8,8,0,711,719)
v0712com primitive to branch on false condition
v0713com list=value,jump-loc: value-prec,:storage,time,ext,calc,incl,addr
v0714begin stext
v07151d a,((val)) ;4m 13t 3b branch to (loc) if (val) is true
v0716cp 0         ;2m  7t 2b
v0717jp z, (loc)  ;3m 10t 3b
v0718endtext
v0719calc romptr=romptr+8
v0720s.monitor   (::1,4,1,7,0,720,727)
v0721com primitive to define p2 monitor as controller supervisor
v0722com list = empty:empty: storage,time,ext,calc,int,addr
v0723begin stext
v0724;          =monitor section=
v0725@spvsr:nop     ;1m    4t 1b mark top of the polling loop
v0726endtext
v0727calc romptr=romptr+1
v0728s.tabend    ( :: 3,10,3,6,0,728,734)
v0729com subroutine to define end of monitor table
v0730com list= empty:empty:s,t,e,c,i,addr)
v0731begin stext
v0732jp  @spvsr     ;go to the top of the polling loop of monitor table
v0733endtext
v0734calc romptr=romptr+3
v0735s.mult      (rslt,arg1,arg2:0,8,0,8,0,8:35,528,138,22,0,735,757)
v0736com binary multiplication primitive
v0737begin stext
v07381d  a ,((arg1));3m    13t    3b     put arg1 in e
v07391d  e,  a     ;1m    4t     1b
v07401d   a,((arg2));3m    13t    3b     load arg2
v07411d  hl, 0     ;3m    10t    3b     clear rslt
v07421d  d,  h     ;1m    4t     1b     clear d for shifts
v07431d   b, 7     ;2m    7t     2b     set counter to 7bits
v0744rra           ;1m    4t     1b
v0745jp  nc,$+4    ;3m    10t    3b
v0746add hl, de    ;3m    11t    1b
v0747sla e         ;2m    8t     2b
v0748rl  d         ;2m    8t     2b
v0749djnz $-9      ;3m    13t    2b +7 +2m 8t on last time
v0750rra           ;1m    4t     1b
v0751jp  nc, $+6   ;3m    10t    3b
v0752and a         ;1m    4t     1b
v0753sbc hl, de    ;4m    15t    2b
v0754ld  a,  l     ;1m    4t     1b     truncate result to 8 bits
```

```
v0755ld  ((rslt)),a ;4m    13t    3b     save result
v0756endtext
v0757calc romptr=romptr+35
v0758s.mult       (rslt,arg1,arg2:0,16,0,8,0,8:34,527,138,21,0,758,779)
v0759com multiply 2 8 bit number and get 16 bit result
v0760begin stext
v0761ld  a ,((arg1));3m    13t    3b     put arg1 in e
v0762ld  e,  a      ;1m    4t     1b
v0763ld   a,((arg2));3m    13t    3b     load arg2
v0764ld  hl, 0      ;3m    10t    3b     clear rslt
v0765ld  d,  h      ;1m    4t     1b     clear d for shifts
v0766ld   b, 7      ;2m    7t     2b     set counter to 7bits
v0767rra            ;1m    4t     1b
v0768jp  nc,$+4     ;3m    10t    3b
v0769add hl, de     ;3m    11t    1b
v0770sla e          ;2m    8t     2b
v0771rl  d          ;2m    8t     2b
v0772djnz $-9       ;3m    13t    2b +7 +2m 8t on last time
v0773rra            ;1m    4t     1b
v0774jp  nc, $+6    ;3m    10t    3b
v0775and a          ;1m    4t     1b
v0776sbc hl, de     ;4m    15t    2b
v0777ld  ((rslt)),hl;5m    16t    3b     save result
v0778endtext
v0779calc romptr=romptr+34
v0780s.mult       (rslt,arg1,arg2:0,16,0,16,0,16:39,1105,289,22,0,780,802)
v0781com multiply 2 16 bit numbers and get 16 bit result
v0782begin stext
v0783ld  de,((arg1));6m    20t    4b     put arg1 in de
v0784ld  bc,((arg2));6m    20t    4b     load arg2
v0785ld  a,  b      ;1m    4t     1b     split arg2 to a&c
v0786ld  hl, 0      ;3m    10t    3b     clear rslt
v0787ld   b, 15d    ;2m    7t     2b     set counter to 7bits
v0788rra            ;1m    4t     1b
v0789rr  c          ;2m    8t     2b
v0790jp  nc, $+4    ;3m    10t    3b
v0791add hl, de     ;3m    11t    1b
v0792sla e          ;2m    8t     2b
v0793rl  d          ;2m    8t     2b
v0794djnz $-00bh    ;3m    13t    2b +7 +2m 8t on last time
v0795rra            ;1m    4t     1b
v0796rr  c          ;2m    8t     2b
v0797jp  nc, $+6    ;3m    10t    3b
v0798and a          ;1m    4t     1b
v0799sbc hl, de     ;4m    15t    2b
v0800ld  ((rslt)),hl;5m    16t    3b     save result
v0801endtext
v0802calc romptr=romptr+39
v0803s.whilestart(top,lpct::1,4,1,6,0,803,814)
v0804com primitive to establish label for top of a while-do loop
v0805com condition to be tested immediately follows this label
```

```
v0806com reps is global variable used to account for timing during
v0807com multiple loops
v0808com lpct is max loop count supplied by designer
v0809calc push reps
v0810calc reps=(lpct)
v0811begin stext
v0812(top):nop      ;1m   4t    1b   top of while-do-loop
v0813endtext
v0814calc romptr=romptr+1
v0815s.whilecon  (rslt,bot:0,8:7,27,8,8,0,815,823)
v0816com primitive to decide whether to jump out of while-do loop based
v0817com on boolean value passed to rslt    bot is loop bottom label
v0818begin stext
v0819ld a,((rslt))   ;4m   13t    3b  get boolean value
v0820and a           ;1m    4t    1b  check if true(ffh) or false(00h)
v0821jp z,(bot)+3    ;3m   10t    3b  if false jump out of while-do loop
v0822endtext
v0823calc romptr=romptr+7
v0824s.whend     (top,bot::3,10,3,4,0,824,832)
v0825com primitive to mark end of statements to be executed in a while-
v0826com do-loop       global variable reps is reset to value existing
v0827com before for-loop started
v0828calc pop reps
v0829begin stext
v0830(bot):jp (top)   ;3m   10t    3b  jump to top of while-do loop
v0831endtext
v0832calc romptr=romptr+3
v0833s.ne        (rslt,arg1,arg2:0,8,0,16,0,16:18,91,26,13,0,833,846)
v0834com primitive to perform comparision between 2 16-bit numbers
v0835com list=result,argument 1, argument 2 ::stor,time,ext,c,i,addrs
v0836begin stext
v0837ld de,((arg1)) ;6m 20t 4b if arg1 = arg2 then rslt=ffh  de=(arg1)
v0838ld hl,((arg2)) ;5m 16t 3b                              hl=(arg2)
v0839and a          ;1m  4t 1b reset carry flag
v0840sbc hl,de      ;4m 15t 2b
v0841ld  a,0        ;2m  7t 2b
v0842jr  z, $+3     ;3m 12t 2b     result equal
v0843cpl            ;1m  4t 1b     result not equal
v0844ld ((rslt)),a  ;4m 13t 3b
v0845endtext
v0846calc romptr=romptr+18
v0847h.processor (:: , , ,2,3,847,871)
v0848com primitive to include z-80 cpu board 4 mhz
v0849calc slot = slot + 1
v0850incl h.tcardcage (::)
v0851begin htext
v0852    put z-80 cpu board in slot (slot)
v0853    memex high
v0854    set jumpers in the following pattern
v0855       jumper     pattern
v0856         w2        010
```

```
v0857          w3          001
v0858          w4          010
v0859          w5          1
v0860          w6          001
v0861          w7          01
v0862          w8          110
v0863          w9          1111
v0864          w10         1
v0865          w12         101010
v0866          w13         10
v0867       w14         10
v0868          w15         01
v0869     note numbering is from left to right and from top to bottom.
v0870     address space 0000-7fff
v0871endtext
v0872h.cardcage  (:: , , ,0,0,872,876)
v0873com primitive to include card cage and power supply for controller
v0874begin htext
v0875     connect powersupply to card cage
v0876endtext
v0877h.memory     (:: , , ,2,3,877,897)
v0878com primitive to include required memory
v0879calc slot = slot + 1
v0880incl h.tcardcage (::)
v0881if romptr .lt. ramptr skip 5
v0882begin htext
v0883     the program space and the variable space have colided
v0884     you do not have enough memory to execute your program
v0885     your memory is limited to 16k
v0886endtext
v0887begin htext
v0888     put 16k memory board in slot (slot)
v0889     set jumpers in the following pattern
v0890        jumper      pattern
v0891          w1          11111111
v0892          w2          10
v0893          w3          0
v0894          w4          01
v0895          w5          1
v0896      address range for card is 4000-7fff
v0897endtext
v0898h.tcardcage (:: , , ,0,0,898,904)
v0899com primitive to limit the number of slots in card cage to 8
v0900if slot .le. 8 skip 4
v0901begin htext
v0902     you have exceeded the maximum number of allowable slots in the
v0903     card cage. it is limited to 8.
v0904endtext
v0905s.divide     (rslt,arg1,arg2:0,8,0,8,0,8:56,504,129,41,0,905,946)
v0906com routine to divide arg1 by arg2 and store in rslt
v0907com taken from zaks p 137
```

```
v0908begin stext
v0909ld  a,  ((arg1)) ;m4    t13    b3     get dividend
v0910and a            ;1m    4t     1b
v0911ld  h,0          ;2m    7t     2b
v0912jp  p,  $+7      ;3m    10t    3b
v0913cpl              ;1m    4t     1b
v0914inc a            ;1m    4t     1b
v0915ld  h,080h       ;2m    7t     2b
v0916ld  e,  a        ;m1    t4     b1
v0917ld  a,  ((arg2)) ;m4    t13    b3     get divisor
v0918and a            ;1m    4t     1b
v0919jp  p,  $+0bh    ;3m    10t    3b
v0920cpl              ;1m    4t     1b
v0921inc a            ;1m    4t     1b
v0922ld  c,  a        ;1m    4t     1b
v0923ld  a,  080h     ;2m    7t     2b
v0924xor h            ;1m    4t     1b
v0925ld  h,  a        ;1m    4t     1b
v0926ld  a,  c        ;1m    4t     1b
v0927ld  c,  a        ;m1    t4     b1
v0928xor a            ;m1    t4     b1     clear accumulator
v0929ld  b,  8        ;m2    t7     b2     set loop counter
v0930rl  e            ;m2    t8     b2     rotate
v0931rla              ;m1    t4     b1
v0932sub c            ;m1    t4     b1     trial subtract
v0933jr  nc  ,$+3     ;m3    t12    b2     subtract ok
v0934add a,  c        ;m1    t4     b1     restore accum,set cy
v0935djnz  $-7        ;m3    t13    b2     m2 t8 on last loop
v0936ld  b,  a        ;m1    t4     b1     put remainder in b
v0937ld  a,  e        ;m1    t4     b1     get quitent
v0938rla              ;m1    t4     b1     shift in last result bit
v0939cpl              ;m1    t4     b1     complement bits
v0940bit 7, h         ;2m    8t     2b
v0941jp  z,$+5        ;3m    10t    3b
v0942cpl              ;1m    4t     1b
v0943inc a            ;1m    4t     1b
v0944ld  ((rslt)),a ;m4    t13    b3     store quotient in rslt
v0945endtext
v0946calc romptr=romptr+56
v0947s.divide    (rslt,arg1,arg2:0,16,0,16,0,16:80,1465,376,57,0,947,1004)
v0948com primitive to divide arg1 by arg2 and store in rslt
v0949com list=rslt,arg1,arg2:precisions:s,t,e,c,i,addr
v0950begin stext
v0951ld hl, ((arg1));5m    16t    3b     load arg1 in hl pair
v0952bit 7, h         ;2m    8t     2b
v0953ld b,0           ;2m    7t     2b
v0954jp z, $+12       ;3m    10t    3b
v0955ld a, h          ;1m    4t     1b
v0956cpl              ;1m    4t     1b
v0957ld h, a          ;1m    4t     1b
v0958ld a, l          ;1m    4t     1b
```

```
v0959cpl            ;1m    4t     1b
v0960ld l, a        ;1m    4t     1b
v0961inc hl         ;1m    6t     1b
v0962ld b, 080h     ;2m    7t     2b
v0963ld de, ((arg2));6m    20t    4b     load arg2 in bc pair
v0964bit 7, d       ;2m    8t     2b
v0965ld a, 0        ;2m    7t     2b
v0966jp z, $+12     ;3m    10t    3b
v0967ld a, d        ;1m    4t     1b
v0968cpl            ;1m    4t     1b
v0969ld d, a        ;1m    4t     1b
v0970ld a, e        ;1m    4t     1b
v0971cpl            ;1m    4t     1b
v0972ld e, a        ;1m    4t     1b
v0973inc de         ;1m    6t     1b
v0974ld a, 080h     ;2m    7t     2b
v0975xor b          ;1m    4t     1b
v0976ex af,af'      ;1m    4t     1b     save sign of rslt
v0977ld c,l         ;1m    4t     1b
v0978ld a,h         ;1m    4t     1b
v0979ld b, 16d      ;2m    7t     2b
v0980ld hl, 0       ;3m    10t    3b
v0981rl c           ;2m    8t     2b     loop
v0982rla            ;1m    4t     1b
v0983adc hl, hl     ;4m    15t    2b
v0984sbc hl,de      ;4m    15t    2b
v0985jr  nc, $+3    ;3m    12t    2b     sub was ok
v0986add hl, de     ;3m    11t    1b     restore accumulator
v0987ccf            ;1m    4t     1b     calc result bit
v0988djnz  $-11     ;3m    13t    2b     2m 8t on =0
v0989rl  c          ;2m    8t     2b
v0990rla            ;1m    4t     1b
v0991ld  h,  a      ;1m    4t     1b
v0992ld  l,  c      ;1m    4t     1b
v0993ex af,af'      ;1m    4t     1b     restore sign of rslt
v0994jp p, $+10     ;3m    10t    3b
v0995ld a, h        ;1m    4t     1b
v0996cpl            ;1m    4t     1b
v0997ld h, a        ;1m    4t     1b
v0998ld a, l        ;1m    4t     1b
v0999cpl            ;1m    4t     1b
v1000ld l, a        ;1m    4t     1b
v1001inc hl         ;1m    6t     1b
v1002ld  ((rslt)),hl;5m    16t    3b     save result
v1003endtext
v1004calc romptr=romptr+80
v1005h.clock     (:: , , ,0,0,1005,1028)
v1006com primitive to create an clock in the ctc chip of the z80 cpu board
v1007begin htext
v1008     these additional connections on the cpu board are required to
v1009     utilize 2 of the 3 channels of the ctc chip     the ctc chip
```

```
v1010     operates at 2 mhz vice 4 mhz for the z-80a cpu    the following
v1011     jumpers will cause channel 0 to be served by the internal 2mhz
v1012     clock and channel 1 to be served by the output from channel 0
v1013     thus for example if channel 0 is set up to generate a pulse
v1014     every 2000 internal clock cycles and this pulse becomes the input
v1015     clock signal to channel 1  then the net result is channel 1 is a
v1016     downcounter supplied with a 1khz clock signal
v1017    connect j1-20 to j1-12   this connects channel 0 output to channel 1
v1018                             input clock
v1019            j1-15 to j1-16   gate of channel 0 tied to ground so down
v1020                             counter will work
v1021            j1-9  to j1-10   gate of channel 1 tied to ground so down
v1022                             counter will work
v1023     connect on w12
v1024              1-2            internal clock signal supplied to channel 2
v1025              7-8            external clock signal supplied to channel 1
v1026                             (actually the output from channel 0)
v1027              9-10           internal clock signal supplied to channel 0
v1028endtext
v1029h.keydisplay(:: , , ,6,8,1029,1048)
v1030com primitive to add the 7303 keyboard display card   this primitive
v1031com is called by outputport and inputport     the keyboard and
v1032com digital display features are not used   only the rocker swithces
v1033com are used to control input and the leds to display output
v1034if keybrd .eq. 1 skip 14
v1035calc keybrd = 1
v1036calc slot = slot + 1
v1037incl h.tcardcage (::)
v1038begin htext
v1039    put first prolog std 7303 keyboard/display card in slot (slot)
v1040    connect the following jumper pins
v1041      x6
v1042      y4
v1043      z0
v1044      z1
v1045    disconnect the following jumper pins
v1046      all others
v1047    address space 11000000, 11000001
v1048endtext
v1049s.fixedwait (time:0,1275:15,-5,18,6,0,1049,1068)
v1050com routine to delay a fixed period of time in increments of 5ms
v1051com max allowed input value is 1275ms
v1052com as currently coded there may be up to a 10% error in actual
v1053com elapsed time when compared to the input value
v1054attr time =(time)*4000
v1055calc scrtch =(time)/5
v1056begin stext
v1057;
v1058; wait (time) ms  (for z80a 4 mhz clock)
v1059;
v1060ld b,(scrtch)     ;2m   7t   2b  set value of outer loop counter
```

```
v1061ld de,-1          ;3m   10t  3b  value by which inner loop is decremntd
v1062ld hl,800         ;3m   10t  3b  starting couter value for inner loop
v1063add hl,de         ;3m   11t  1b  decrement inner loop
v1064nop               ;1m   4t   1b  dummy inst. to make inner loop =25t
v1065jp c,$-2          ;3m  10t   3b  jump to of inner loop until hl=0
v1066djnz $-8          ;3m2 13t8  2b  decrement outer loop counter until b=0
v1067endtext
v1068calc     romptr=romptr + 15
v1069s.issuevent (outnm:0,8:5,24,7,8,0,1069,1077)
v1070com outputs contents of outnm to data port of prolog 7303 keyboard
v1071com card   data port is d0h     value of data sent can be seen by
v1072com examining 8 leds on 7303 card, one led for each of 8 bits
v1073begin stext
v1074ld a,((outnm))  ;4m  13t  3b  get contents of output variable
v1075out (0d0h),a      ;3m  11t  2b  output to data port of 7303 card
v1076endtext
v1077calc     romptr=romptr + 5
v1078s.outputport(outnm,tech:0,8:6,29,8,14,13,1078,1103)
v1079com  tech is a hold-over from the original csde design
v1080com it is not used here because the output type of signal
v1081com is predetermined by the hardware available, prolog boards
v1082com keybrd is a boolean flag indicating if the prolog 7303 board has
v1083com been included already   this primitive sets up the 7303 card
v1084com so that contents of outnm will be output to the single data
v1085com port, d0h    to do this the variable must first be created  then a
v1086com control code sent to port d1h to write inhibit the digit displays
v1087com any data value that is output will be seen only on the 8 leds
v1088com on=1 and off=0 for each of 8 bits of the output data value
v1089com the leds are cleared first in preparation for display of new data
v1090if keybrd .ne. 0 skip 2
v1091incl h.keydisplay(::)
v1092calc keybrd = keybrd + 1
v1093calc ramptr = ramptr - 1
v1094begin stext
v1095;sets up 7303 card so that the contents of (outnm) will be output
v1096org (ramptr)
v1097(outnm): defb 0
v1098org (romptr)
v1099ld a, 0        ; 2m   7t    2b   write inhibit the alphanumeric display
v1100out (0d1h),a   ; 3m  11t    2b   send it to control port
v1101out (0d0h),a   ; 3m  11t    2b   clear all leds
v1102endtext
v1103calc     romptr = romptr + 6
v1104s.inputport (innam,tech:0,8:0,0,0,13,12,1104,1125)
v1105com tech is a hold-over from the original csde design  it is not used
v1106com here because the input type of signal is predetermined by the
v1107com 7303 keyboard/display board ie, a single 8-bit data port    since
v1108com no control code is required, only the input storage location is
v1109com created by this primitive    when more complex i/o hardware is
v1110com available this primitive will require modification
v1111com innam is where the value available at the single data port, d0h,
```

```
v1112com will be latched
v1113com keybrd is a boolean flag indicating if the prolog 7303 card has
v1114com already been included
v1115if keybrd .ne. 0 skip 2
v1116incl h.keydisplay(::)
v1117calc keybrd = keybrd +1
v1118calc ramptr = ramptr -1
v1119begin stext
v1120; sets up 7303 card so that value at data port can be
v1121; read into (innam) by the primitive s.sensecond
v1122org (ramptr)
v1123(innam):  defb 0
v1124org (romptr)
v1125endtext
v1126s.sensecond (innam:0,8:56,129,37,44,0,1126,1170)
v1127com purpose is to demonstrate ability to input data
v1128com innam is the variable that would normally be the depository
v1129com of the value present at the single data port , d0h, on the
v1130com 7303 keyboard card     for demonstration purposes only the 2
v1131com rocker switches on the 7303 card are used to control input
v1132com and since they only control bits 6 and 7 of the 8 bit(0-7)
v1133com data port, a small conversion routine has been added such that
v1134com 1 of 4 values will be placed in  innam  depending on the
v1135com positions of the 2 rocker switches     the following table
v1136com applies        s2(left)      s1(right)      value put in innam
v1137com                 on(up)         on                04h
v1138com                 on             off(down)         03h
v1139com                off             on                02h
v1140com                off             off               01h
v1141com  this allows an input choice of 4 differnt values via the
v1142com  pro-log rocker switches on the 7303 keyboard display card
v1143begin stext
v1144in a,(0d0h)          ;  3m   11t   2b   data port read for input
v11451d b,a               ;  1m    4t   1b   save value in b for later
v1146and 11000000b        ;  2m    7t   2b   mask for both switches on
v1147cp 11000000b         ;  2m    7t   2b   check for both switches on
v1148jp z,$+27            ;  3m   10t   3b   if both on then jump down
v11491d a,b               ;  1m    4t   1b   get original value again
v1150and 10000000b        ;  2m    7t   2b   mask for left switch on only
v1151cp 10000000b         ;  2m    7t   2b   check for left switch on only
v1152jp z,$+27            ;  3m   10t   3b   if left on then jump down
v11531d a,b               ;  1m    4t   1b   get original value again
v1154and 01000000b        ;  2m    7t   2b   mask for right switch on only
v1155cp 01000000b         ;  2m    7t   2b   check for rt switch on only
v1156jp z,$+27            ;  3m   10t   3b   if rt on then jump down
v11571d a,1               ;  2m    7t   2b   both switches must be off
v11581d ((innam)),a       ;  4m   13t   3b   both off, (innam) = 01h
v1159jp $+24              ;  3m   10t   3b   jump to end of routine
v11601d a,4               ;  2m    7t   2b   both switches must be on
v11611d ((innam)),a       ;  4m   13t   3b   both on, (innam) = 04h
v1162jp $+16              ;  3m   10t   3b   jump to end of routine
```

```
v1163ld a,3              ; 2m   7t   2b  left switch on only
v1164ld ((innam)),a      ; 4m  13t   3b  left on, (innam) = 03h
v1165jp $+8              ; 3m  10t   3b  jump to end of routine
v1166ld a,2              ; 2m   7t   2b  right switch on only
v1167ld ((innam)),a      ; 4m  13t   3b  right on, (innam) = 02h
v1168nop                 ; 1m   4t   1b  end of input conversion routine
v1169endtext
v1170calc      romptr=romptr + 56
v1171s.stboolwait(top,maxtm::1,-5,1,10,6,1171,1181)
v1172com primitive to mark top of boolean wait structure
v1173com top is label for beginning of boolean wait
v1174com maxtm is max time in milliseconds allowed to check conditions
v1175com between s.stboolwait and s.boolwait
v1176attr time=(maxtm)*4000
v1177call s.setime    ((maxtm):)
v1178begin stext
v1179(top):nop            ;1m   4t   1b      mark top of boolean wait loop
v1180endtext
v1181calc romptr=romptr+1
v1182s.boolwait  (rslt,top,bot:0,8:22,93,26,23,0,1182,1205)
v1183com primitive to check for boolean condition(if true then exit) and
v1184com read current time from channel 1 of ctc       since clock
v1185com continues to downcount past 0000h  time interval expiration
v1186com is determined by checking the sign bit of the msb of the 2 byte
v1187com clock time     if it is 1 then time has expired and the boolean
v1188com structure is exited        rslt  is boolean value passed from
v1189com condition being checked   top and bot are labels
v1190begin stext
v1191ld a,((rslt))       ;4m   13t   3b     get boolean value
v1192and a               ;1m    4t   1b     check if true(ffh) or false(00h)
v1193jp nz,(bot)+3        ;3m   10t   3b     if true jump out
v1194ld a,01000001b      ;2m    7t   2b     channel1+latched read+mode0+hex
v1195out (0f3h),a        ;3m   11t   2b     send to control code port
v1196in a,(0f1h)         ;3m   11t   2b     read lsb
v1197ld l,a              ;1m    4t   1b     save lsb
v1198in a,(0f1h)         ;3m   11t   2b     read msb
v1199ld h,a              ;1m    4t   1b     save msb
v1200bit 7,h             ;2m    8t   2b     check if counter value has passed
v1201                    ;                  zero  ie, become negative
v1202(bot):jp z,(top)    ;3m    10t  3b     if counter value still positive
v1203                    ;                  ie, bit 7 = 0 then go to top
v1204endtext
v1205calc romptr=romptr+22
v1206s. call     (nam ::3,17,5,7,0,1206,1213)
v1207com  primitive to call another procedure
v1208com  list=proc-name:empty:storage,time,ext,calc,incl,addr
v1209begin stext
v1210; call procedure (nam)
v1211call @(nam)          ; 5m   17t   3b
v1212endtext
v1213calc romptr=romptr+3
```

```
v1214s.in       (:: , , ,9,0,1214,1223)
v1215com primitive to set the timed block flag
v1216com it is modeled exactly after ltcol ross's s.in in the 8080
v1217com realization volumn  and is included for completeness
v1218com it does not conform to carson's translator output format
v1219com and is not usable in its present form
v1220com the global variable tmblck is supposed to be a flag to indicate
v1221com to the csde program that the following primitives constitute
v1222com a timed block within a task
v1223calc tmblck=1
v1224s.ni       (:: , , ,3,0,1224,1227)
v1225com primitive to clear the timed block flag
v1226com same comments as in s.in apply
v1227calc tmblck=0
v1228h.uart     (:: , , ,0,0,1228,1242)
v1229begin htext
v1230  this is a dummy primitive to remind you to put in the dual uart card
v1231  if you wish to use the nps loading rom.  the require setting are as
v1232  follows.
v1233    set jumpers in the following pattern
v1234       jumper     pattern
v1235         w1        01
v1236         w2        01
v1237         w3        10
v1238         sx        0001
v1239         sy        00001000
v1240    address space e0 thru e7
v1241endtext
v1242com this has to be the last line
```

## APPENDIX F

## WORKING CSDL TEST PROGRAM

This appendix contains a problem to test the primitives that generate a while-do loop. This problem was completely run through CSDE, from CSDL problem statement to operating program on the Pro-log microcomputer. The files listed below are unedited except for the primitive list, where the value of max loop count was moved from 's.whilecon' to 's.whilestart'. They are, in order, the CSDL problem, primitive list, application timing file, symbol table, CSDE software output, CSDE hardware output, and CSDE debug file.

### CSDL Problem

```
IDENTIFICATION
      DESIGNER : "BOB VOGEL"
      DATE : "02-07-85"
      PROJECT : "WHILE DO CONTRUCT TEST"


DESIGN CRITERIA
       METRIC FIRST;
       VOLUMES 1;
       MONITORS 1;


ENVIRONMENT

INPUT: ARG1,8,TTL; END INPUT;

OUTPUT: LIGHT,8,TTL; END OUTPUT;

ARITHMETIC: EACH1,8; EACH5,8;
END ARITHMETIC;

PROCEDURES


FUNCTION EACH1:
       BINARY,1;
       EACH1:=0;
       SENSE (ARG1);
       IF ARG1<=2 THEN EACH1:=-1; END IF;
END EACH1;
```

```
FUNCTION EACH5:
        BINARY,1;
        EACH5:=0;
        SENSE (ARG1);
        IF ARG1<=4 THEN EACH5:=-1; END IF;
END EACH5;

TASK ONLITA;
        LIGHT:=1;
        WHILE LIGHT <= 7 : 4 DO
              ISSUE (LIGHT);
              LIGHT:=LIGHT + 2;
              WAIT 250MS;
        END WHILE;
END ONLITA;


TASK OFFLT;
        LIGHT:=0;  ISSUE (LIGHT);
        WAIT 500MS;
END OFFLT;


CONTINGENCY LIST
     WHEN EACH1 : 1600MS DO ONLITA;
     WHEN EACH5 : 1600MS DO OFFLT;
END
```

Primitive list

```
P   1t.generated for:    SYSTEM              ******************
P   2s.MAIN       (::)
P   3d:FIRST      :          1:          1:
P   4s.inputport  (ARG1,TTL:8)
P   5s.outputport(LIGHT,TTL:8)
P   6s.var        (EACH1:8,0)
P   7s.var        (EACH5:8,0)
P   8t.generated for: EACH1                  ******************
P   9s.proc       (EACH1:)
P  10s.assign     (EACH1,@C01:1,8)
P  11s.sensecond  (ARG1:8)
P  12s.le         (@T01,ARG1,@C02:8,8,8)
P  13s.jmpf       (@T01,@01:8)
P  14s.sub        (@T01,@C01,@C03:8,8,8)
P  15s.assign     (EACH1,@T01:1,8)
P  16s.loc        (@01:)
P  17s.exitproc   (EACH1:)
P  18t.generated for: EACH5                  ******************
```

```
P  19s.proc        (EACH5:)
P  20s.assign      (EACH5,@C01:1,8)
P  21s.sensecond   (ARG1:8)
P  22s.le          (@T01,ARG1,@C04:8,8,8)
P  23s.jmpf        (@T01,@02:8)
P  24s.sub         (@T01,@C01,@C03:8,8,8)
P  25s.assign      (EACH5,@T01:1,8)
P  26s.loc         (@02:)
P  27s.exitproc    (EACH5:)
P  28t.generated for: ONLITA                  ****************
P  29s.proc        (ONLITA:)
P  30s.assign      (LIGHT,@C03:8,8)
P  31s.whilestart(@03,4:)
P  32s.le          (@T01,LIGHT,@C05:8,8,8)
P  33s.whilecon    (@T01,@04:8)
P  34s.issuevent   (LIGHT:8)
P  35s.add         (@T01,LIGHT,@C02:8,8,8)
P  36s.assign      (LIGHT,@T01:8,8)
P  37s.fixedwait   (250)
P  38s.whend       (@03,@04:)
P  39s.exitproc    (ONLITA:)
P  40t.generated for: OFFLT                   ****************
P  41s.proc        (OFFLT:)
P  42s.assign      (LIGHT,@C01:8,8)
P  43s.issuevent   (LIGHT:8)
P  44s.fixedwait   (500)
P  45s.exitproc    (OFFLT:)
P  46t.generated for:   SYSTEM                ****************
P  47s.cons        (@C01,0:8)
P  48s.cons        (@C02,2:8)
P  49s.cons        (@C03,1:8)
P  50s.cons        (@C04,4:8)
P  51s.cons        (@C05,7:8)
P  52s.var         (@T01:8)
```

Application timing file

```
A   1 :EACH1      :ONLITA     :MS:1600,   0,   0,   0,   0
A   2 :EACH5      :OFFLT      :MS:1600,   0,   0,   0,   0
```

Symbol table

```
S. INPUTPORT(ARG1,TTL:8)
S.OUTPUTPORT(LIGHT,TTL:8)
S.VARIABLE   (EACH1:8,0)
S.VARIABLE   (EACH5:8,0)
S.LOC        (@01:)
S.LOC        (@02:)
```

```
S.CONS      (@CO1:0,8)
S.CONS      (@CO2:2,8)
S.CONS      (@CO3:1,8)
S.CONS      (@CO4:4,8)
S.CONS      (@CO5:7,8)
```

## Software output

```
;                                                                    ;  1
;          zilog z-80 based system                                   ;  2
;                                                                    ;  3
;                                                                    ;  4
;                                                                    ;  5
;                                                                    ;  6
;                                                                    ;  7
;                                                                    ;  8
.z80                                                                 ;  9
aseg                                                                 ; 10
org 32735          ;ram pointer is pointing to top of memory - stack ; 11
@stak:defs 32              ;            32b    define stack area     ; 12
org   16384        ;begin code after reserved interrupt area         ; 13
@cold:ld  sp,@stak+32      ;3m    10t    3b     initialize stack point; 14
di                    ;1m    4t     1b     disable maskable interrupts; 15
jp @i0         ;3m    10t    3b     do hardware initializations     ; 16
; sets up 7303 card so that value at data port can be               ; 17
; read into arg1 by the primitive s.sensecond                       ; 18
org 32734                                                            ; 19
arg1:  defb 0                                                        ; 20
org 16391                                                            ; 21
;sets up 7303 card so that the contents of light will be output     ; 22
org 32733                                                            ; 23
light: defb 0                                                        ; 24
org 16391                                                            ; 25
ld a, 0        ; 2m   7t    2b   write inhibit the alphanumeric displa; 26
out (0d1h),a   ; 3m  11t    2b   send it to control port             ; 27
out (0d0h),a   ; 3m  11t    2b   clear all leds                      ; 28
org  32732       ;8 bit variable each1 in ram                       ; 29
each1:  defb  0    ;0m    0t     1b                                  ; 30
org  16397                                                           ; 31
org  32731       ;8 bit variable each5 in ram                       ; 32
each5:  defb  0    ;0m    0t     1b                                  ; 33
org  16397                                                           ; 34
;procedure each1                                                    ; 35
@each1: nop    ;1m  4t     1b entry point for each1                  ; 36
ld  a,(@c01) ;4m    13t    3b     assign @c01                        ; 37
ld  (each1),a  ;4m    13t    3b          to     each1                ; 38
in a,(0d0h)        ;  3m   11t   2b   data port read for input       ; 39
ld b,a             ;  1m    4t   1b   save value in b for later      ; 40
and 11000000b      ;  2m    7t   2b   mask for both switches on      ; 41
```

```
cp 11000000b       ;  2m    7t    2b   check for both switches on      ;  42
jp z,$+27          ;  3m   10t    3b   if both on then jump down      ;  43
ld a,b             ;  1m    4t    1b   get original value again      ;  44
and 10000000b      ;  2m    7t    2b   mask for left switch on only  ;  45
cp 10000000b       ;  2m    7t    2b   check for left switch on only ;  46
jp z,$+27          ;  3m   10t    3b   if left on then jump down     ;  47
ld a,b             ;  1m    4t    1b   get original value again      ;  48
and 01000000b      ;  2m    7t    2b   mask for right switch on only ;  49
cp 01000000b       ;  2m    7t    2b   check for rt switch on only   ;  50
jp z,$+27          ;  3m   10t    3b   if rt on then jump down       ;  51
ld a,1             ;  2m    7t    2b   both switches must be off     ;  52
ld (arg1),a     ;  4m   13t    3b   both off, arg1 = 01h             ;  53
jp $+24            ;  3m   10t    3b   jump to end of routine        ;  54
ld a,4             ;  2m    7t    2b   both switches must be on      ;  55
ld (arg1),a     ;  4m   13t    3b   both on, arg1 = 04h              ;  56
jp $+16            ;  3m   10t    3b   jump to end of routine        ;  57
ld a,3             ;  2m    7t    2b   left switch on only           ;  58
ld (arg1),a     ;  4m   13t    3b   left on, arg1 = 03h              ;  59
jp $+8             ;  3m   10t    3b   jump to end of routine        ;  60
ld a,2             ;  2m    7t    2b   right switch on only          ;  61
ld (arg1),a     ;  4m   13t    3b   right on, arg1 = 02h             ;  62
nop                ;  1m    4t    1b   end of input conversion routine ;  63
ld  a,(arg1) ;4m    13t    3b     if arg1 le arg2 then rslt=ffh      ;  64
ld  b,  a      ;1m    4t     1b     b=arg1                           ;  65
ld  a,(@c02) ;4m    13t    3b                                      ;  66
ld  c,  a      ;1m    4t     1b     c=arg2                           ;  67
and a          ;1m    4t     1b     set sign flag of arg2            ;  68
jp  p,$+13     ;3m    10t    3b     jump if arg2 is positive         ;  69
ld  a,  b      ;1m    4t     1b     arg2 = -                         ;  70
and a          ;1m    4t     1b     set sign flag of arg1            ;  71
ld  a,  c      ;1m    4t     1b     restore arg2 to accumulator      ;  72
jp  m,  $+17   ;3m    10t    3b     arg1 = - arg2 = - comp backwards ;  73
ld  a,  0      ;2m    7t     2b     arg1 = + arg2 = - false          ;  74
jr  $+13       ;3m    12t    2b                                      ;  75
ld  a,  b      ;1m    4t     1b                                      ;  76
and a          ;1m    4t     1b     set sign flag of arg1            ;  77
ld  a,  c      ;1m    4t     1b     restore arg2 to accumulator      ;  78
jp  p,  $+7    ;3m    10t    3b     arg1 = + arg2 = +                ;  79
ld  a,11111111b;2m    7t     2b     arg1 = - arg2 = + true           ;  80
jr  $+9        ;3m    12t    2b                                      ;  81
cp  b          ;1m    4t     1b                                      ;  82
ld  a,11111111b;2m    7t     2b     result false arg1 )= arg2        ;  83
jp  p,  $+4    ;3m    10t    3b                                      ;  84
cpl            ;1m    4t     1b     result true arg1 lt  arg2        ;  85
ld  (@t01),a ;4m    13t    3b                                      ;  86
ld a,(@t01) ;4m 13t 3b branch to @01 if @t01 is true               ;  87
cp 0         ;2m  7t 2b                                            ;  88
jp z, @01  ;3m 10t 3b                                             ;  89
ld a,(@c01)  ;4m    13t    3b     load arg1 in accumulator         ;  90
ld hl,@c03   ;3m    10t    3b     point hl to arg2                 ;  91
sub (hl)       ;2m     7t    1b     arg1 - arg2                      ;  92
```

```
jp po ,$+13    ;3m   10t 3b  if no overflow store result                ;  93
jp c  ,$+8     ;3m   10t 3b  if carry the maximize minus rslt            ;  94
ld  a,01111111b;2m    7t 2b  put in largest positive value             ;  95
jp $+5         ;3m   10t 3b                                            ;  96
ld a,10000000b ;2m    7t 2b  put in largest negative value             ;  97
ld (@t01),a  ;13t   4m 3b  save result of add in rslt                   ;  98
ld  a,(@t01) ;4m    13t    3b     assign @t01                           ;  99
ld  (each1),a  ;4m    13t    3b           to     each1                  ; 100
@01:  nop                  ; define location @01                       ; 101
ret            ;3m    10t    1b     return to monitor,exit each1      ; 102
;procedure each5                                                       ; 103
@each5: nop    ;1m  4t     1b entry point for each5                    ; 104
ld  a,(@c01) ;4m    13t    3b     assign @c01                           ; 105
ld  (each5),a  ;4m    13t    3b           to     each5                  ; 106
in a,(0d0h)        ;  3m   11t   2b   data port read for input         ; 107
ld b,a             ;  1m    4t   1b   save value in b for later       ; 108
and 11000000b      ;  2m    7t   2b   mask for both switches on       ; 109
cp 11000000b       ;  2m    7t   2b   check for both switches on      ; 110
jp z,$+27          ;  3m   10t   3b   if both on then jump down      ; 111
ld a,b             ;  1m    4t   1b   get original value again        ; 112
and 10000000b      ;  2m    7t   2b   mask for left switch on only   ; 113
cp 10000000b       ;  2m    7t   2b   check for left switch on only  ; 114
jp z,$+27          ;  3m   10t   3b   if left on then jump down      ; 115
ld a,b             ;  1m    4t   1b   get original value again        ; 116
and 01000000b      ;  2m    7t   2b   mask for right switch on only  ; 117
cp 01000000b       ;  2m    7t   2b   check for rt switch on only     ; 118
jp z,$+27          ;  3m   10t   3b   if rt on then jump down        ; 119
ld a,1             ;  2m    7t   2b   both switches must be off       ; 120
ld (arg1),a     ;  4m   13t   3b   both off, arg1 = 01h               ; 121
jp $+24            ;  3m   10t   3b   jump to end of routine         ; 122
ld a,4             ;  2m    7t   2b   both switches must be on        ; 123
ld (arg1),a     ;  4m   13t   3b   both on, arg1 = 04h               ; 124
jp $+16            ;  3m   10t   3b   jump to end of routine         ; 125
ld a,3             ;  2m    7t   2b   left switch on only            ; 126
ld (arg1),a     ;  4m   13t   3b   left on, arg1 = 03h               ; 127
jp $+8             ;  3m   10t   3b   jump to end of routine         ; 128
ld a,2             ;  2m    7t   2b   right switch on only           ; 129
ld (arg1),a     ;  4m   13t   3b   right on, arg1 = 02h              ; 130
nop                ;  1m    4t   1b   end of input conversion routine ; 131
ld  a,(arg1) ;4m    13t    3b     if arg1 le arg2 then rslt=ffh       ; 132
ld  b,  a      ;1m    4t     1b     b=arg1                            ; 133
ld  a,(@c04) ;4m    13t    3b                                       ; 134
ld  c,  a      ;1m    4t     1b     c=arg2                            ; 135
and a          ;1m    4t     1b     set sign flag of arg2             ; 136
jp  p,$+13     ;3m    10t    3b     jump if arg2 is positive          ; 137
ld  a,  b      ;1m    4t     1b     arg2 = -                          ; 138
and a          ;1m    4t     1b     set sign flag of arg1             ; 139
ld  a,  c      ;1m    4t     1b     restore arg2 to accumulator       ; 140
jp  m,  $+17   ;3m    10t    3b     arg1 = - arg2 = - comp backwards  ; 141
ld  a,  0      ;2m    7t     2b     arg1 = + arg2 = - false           ; 142
jr  $+13       ;3m    12t    2b                                       ; 143
```

```
ld  a,  b       ;1m    4t     1b                                        ; 144
and a           ;1m    4t     1b     set sign flag of arg1              ; 145
ld  a,  c       ;1m    4t     1b     restore arg2 to accumulator        ; 146
jp  p,  $+7     ;3m    10t    3b     arg1 = + arg2 = +                  ; 147
ld  a,11111111b;2m     7t     2b     arg1 = - arg2 = + true             ; 148
jr  $+9         ;3m    12t    2b                                        ; 149
cp  b           ;1m    4t     1b                                        ; 150
ld  a,11111111b;2m     7t     2b     result false arg1 >= arg2          ; 151
jp  p,  $+4     ;3m    10t    3b                                        ; 152
cpl             ;1m    4t     1b     result true arg1 lt  arg2          ; 153
ld  (@t01),a ;4m    13t    3b                                           ; 154
ld a,(@t01) ;4m 13t 3b branch to @02 if @t01 is true                    ; 155
cp 0         ;2m  7t 2b                                                 ; 156
jp z, @02  ;3m 10t 3b                                                  ; 157
ld a,(@c01)  ;4m    13t    3b     load arg1 in accumulator               ; 158
ld hl,@c03   ;3m    10t    3b     point hl to arg2                       ; 159
sub (hl)       ;2m     7t    1b     arg1 - arg2                         ; 160
jp po ,$+13    ;3m   10t 3b  if no overflow store result                ; 161
jp c  ,$+8     ;3m   10t 3b  if carry the maximize minus rslt            ; 162
ld  a,01111111b;2m     7t 2b  put in largest positive value            ; 163
jp $+5         ;3m   10t 3b                                             ; 164
ld a,10000000b ;2m     7t 2b  put in largest negative value            ; 165
ld (@t01),a  ;13t   4m 3b  save result of add in rslt                   ; 166
ld  a,(@t01) ;4m    13t    3b     assign @t01                           ; 167
ld  (each5),a  ;4m    13t    3b           to     each5                  ; 168
@02:  nop                  ; define location @02                       ; 169
ret             ;3m    10t    1b     return to monitor,exit each5       ; 170
;procedure onlita                                                       ; 171
@onlita: nop    ;1m  4t     1b entry point for onlita                   ; 172
ld  a,(@c03) ;4m    13t    3b     assign @c03                           ; 173
ld  (light),a  ;4m    13t    3b           to     light                  ; 174
@03:nop      ;1m   4t    1b   top of while-do-loop                      ; 175
ld  a,(light) ;4m    13t    3b     if arg1 le arg2 then rslt=ffh        ; 176
ld  b,  a       ;1m    4t     1b     b=arg1                             ; 177
ld  a,(@c05) ;4m    13t    3b                                           ; 178
ld  c,  a       ;1m    4t     1b     c=arg2                             ; 179
and a           ;1m    4t     1b     set sign flag of arg2              ; 180
jp  p,$+13      ;3m    10t    3b     jump if arg2 is positive           ; 181
ld  a,  b       ;1m    4t     1b     arg2 = -                           ; 182
and a           ;1m    4t     1b     set sign flag of arg1              ; 183
ld  a,  c       ;1m    4t     1b     restore arg2 to accumulator        ; 184
jp  m,  $+17    ;3m    10t    3b     arg1 = - arg2 = - comp backwards   ; 185
ld  a,  0       ;2m    7t     2b     arg1 = + arg2 = - false            ; 186
jr  $+13        ;3m    12t    2b                                        ; 187
ld  a,  b       ;1m    4t     1b                                        ; 188
and a           ;1m    4t     1b     set sign flag of arg1              ; 189
ld  a,  c       ;1m    4t     1b     restore arg2 to accumulator        ; 190
jp  p,  $+7     ;3m    10t    3b     arg1 = + arg2 = +                  ; 191
ld  a,11111111b;2m     7t     2b     arg1 = - arg2 = + true             ; 192
jr  $+9         ;3m    12t    2b                                        ; 193
cp  b           ;1m    4t     1b                                        ; 194
```

```
ld  a,11111111b;2m    7t     2b     result false arg1 )= arg2          ; 195
jp  p,  $+4     ;3m    10t    3b                                        ; 196
cpl             ;1m    4t     1b     result true arg1 lt  arg2          ; 197
ld  (@t01),a ;4m    13t    3b                                          ; 198
ld a,(@t01)   ;4m   13t    3b  get boolean value                       ; 199
and a           ;1m    4t    1b  check if true(ffh) or false(00h)      ; 200
jp z,@04+3    ;3m   10t    3b  if false jump out of while-do loop      ; 201
ld a,(light)  ;4m  13t  3b  get contents of output variable           ; 202
out (0d0h),a      ;3m  11t  2b  output to data port of 7303 card       ; 203
ld a, (light) ;13t   4m 3b  store arg1 in accumulator                 ; 204
ld hl,@c02   ;10t   3m 3b  have hl point to arg2 byte                 ; 205
add a, (hl)    ;7t    2m 1b  add accumulator with arg2                ; 206
jp po ,$+13    ;3m   10t 3b  if no overflow store result              ; 207
jp c  ,$+8     ;3m   10t 3b  if carry the maximize minus rslt          ; 208
ld  a,01111111b;2m    7t 2b  put in largest positive value            ; 209
jp $+5         ;3m   10t 3b                                           ; 210
ld a,10000000b ;2m    7t 2b  put in largest negative value            ; 211
ld (@t01),a  ;13t   4m 3b  save result of add in rslt                 ; 212
ld  a,(@t01) ;4m    13t    3b     assign @t01                         ; 213
ld  (light),a  ;4m    13t    3b           to     light                ; 214
;                                                                     ; 215
; wait 250 ms  (for z80a 4 mhz clock)                                 ; 216
;                                                                     ; 217
ld b,50    ;2m   7t   2b  set value of outer loop counter             ; 218
ld de,-1         ;3m   10t  3b  value by which inner loop is decremntd; 219
ld hl,800        ;3m   10t  3b  starting couter value for inner loop  ; 220
add hl,de        ;3m   11t  1b  decrement inner loop                  ; 221
nop              ;1m   4t   1b  dummy inst. to make inner loop =25t   ; 222
jp c,$-2         ;3m  10t   3b  jump to of inner loop until hl=0      ; 223
djnz $-8         ;3m2 13t8  2b  decrement outer loop counter until b=0; 224
@04:jp @03   ;3m   10t    3b  jump to top of while-do loop            ; 225
ret            ;3m    10t    1b     return to monitor,exit onlita     ; 226
;procedure offlt                                                      ; 227
@offlt: nop    ;1m  4t     1b entry point for offlt                   ; 228
ld  a,(@c01) ;4m    13t    3b     assign @c01                         ; 229
ld  (light),a  ;4m    13t    3b           to     light                ; 230
ld a,(light)  ;4m  13t  3b  get contents of output variable           ; 231
out (0d0h),a      ;3m  11t  2b  output to data port of 7303 card      ; 232
;                                                                     ; 233
; wait 500 ms  (for z80a 4 mhz clock)                                 ; 234
;                                                                     ; 235
ld b,100    ;2m   7t   2b  set value of outer loop counter            ; 236
ld de,-1         ;3m   10t  3b  value by which inner loop is decremntd; 237
ld hl,800        ;3m   10t  3b  starting couter value for inner loop  ; 238
add hl,de        ;3m   11t  1b  decrement inner loop                  ; 239
nop              ;1m   4t   1b  dummy inst. to make inner loop =25t   ; 240
jp c,$-2         ;3m  10t   3b  jump to of inner loop until hl=0      ; 241
djnz $-8         ;3m2 13t8  2b  decrement outer loop counter until b=0; 242
ret            ;3m    10t    1b     return to monitor,exit offlt     ; 243
@c01:   defb  0  ;reserve one byte for data                           ; 244
@c02:   defb  2  ;reserve one byte for data                           ; 245
```

```
@c03:   defb  1  ;reserve one byte for data                          ; 246
@c04:   defb  4  ;reserve one byte for data                          ; 247
@c05:   defb  7  ;reserve one byte for data                          ; 248
org  32730       ;8 bit variable @t01 in ram                        ; 249
@t01:  defb  0    ;0m    0t     1b                                   ; 250
org  16828                                                          ; 251
;          =monitor section=                                        ; 252
@spvsr:nop      ;1m    4t 1b mark top of the polling loop           ; 253
call  @each1  ;5m    17t    3b     test for contingency each1       ; 254
ld  a,(each1)  ;4m    13t    3b     get contingency result          ; 255
cp  11111111b  ;1m    4t     1b     check if result true            ; 256
call z,@onlita ;5m    17t    3b     if true execute task            ; 257
               ;if not true get next tabent or tabend to loop       ; 258
call  @each5  ;5m    17t    3b     test for contingency each5       ; 259
ld  a,(each5)  ;4m    13t    3b     get contingency result          ; 260
cp  11111111b  ;1m    4t     1b     check if result true            ; 261
call z,@offlt ;5m    17t    3b     if true execute task            ; 262
               ;if not true get next tabent or tabend to loop       ; 263
call  @each5  ;5m    17t    3b     test for contingency each5       ; 264
ld  a,(each5)  ;4m    13t    3b     get contingency result          ; 265
cp  11111111b  ;1m    4t     1b     check if result true            ; 266
call z,@offlt ;5m    17t    3b     if true execute task            ; 267
               ;if not true get next tabent or tabend to loop       ; 268
jp  @spvsr     ;go to the top of the polling loop of monitor table  ; 269
    ; this space is deliberately void.  this is a dummy primitive.  ; 270
    ; this space is deliberately void.  this is a dummy primitive.  ; 271
@i0:jp @spvsr ;3m 10t 3b  initialization of hardware is complete    ; 272
                    ;          start top of main monitor loop      ; 273
end                        ;end of software listing ready for assemb; 274
```

## Hardware list

```
CAD80, Version 1.3f, Feb. 8, 1985
    put z-80 cpu board in slot 1                                    ;  1
    memex high                                                      ;  2
    set jumpers in the following pattern                            ;  3
       jumper     pattern                                           ;  4
         w2        010                                             ;  5
         w3        001                                             ;  6
         w4        010                                             ;  7
         w5        1                                               ;  8
         w6        001                                             ;  9
         w7        01                                              ; 10
         w8        110                                             ; 11
         w9        1111                                            ; 12
         w10       1                                               ; 13
         w12       101010                                          ; 14
         w13       10                                              ; 15
      w14       10                                                 ; 16
```

```
        w15      01                                                  ;  17
      note numbering is from left to right and from top to bottom.   ;  18
      address space 0000-7fff                                        ;  19
      connect powersupply to card cage                               ;  20
      put first prolog std 7303 keyboard/display card in slot 2      ;  21
      connect the following jumper pins                              ;  22
        x6                                                           ;  23
        y4                                                           ;  24
        z0                                                           ;  25
        z1                                                           ;  26
      disconnect the following jumper pins                           ;  27
        all others                                                   ;  28
      address space 11000000, 11000001                               ;  29
      put 16k memory board in slot 3                                 ;  30
      set jumpers in the following pattern                           ;  31
         jumper     pattern                                         ;  32
           w1        11111111                                       ;  33
           w2        10                                             ;  34
           w3        0                                              ;  35
           w4        01                                             ;  36
           w5        1                                              ;  37
      address range for card is 4000-7fff                            ;  38
  1 this realization consumes   0.000 watts of power
  contains    0 chips.
```

## Debug file

```
CAD80, Version 1.3f, Feb. 8, 1985
error! funmap detected no colon in primitive
37s.fixedwait (250)
funmap forcing colon into primitive
s.fixedwait (250:)
error! funmap detected no colon in primitive
44s.fixedwait (500)
funmap forcing colon into primitive
s.fixedwait (500:)


   1 this realization consumes   0.000 watts of power
and contains    0 chips.

  2 errors in cad80, result =  0
```

## APPENDIX G

## SUMMARY OF TRANSLATOR ERRORS

This appendix contains a summary of Translator errors. These errors are discussed in detail in section III.C. and section IV.B. If the Translator undergoes maintenance in the future, manual editing of primitive lists can be eliminated during the course of developing controller realizations.

1. Format errors relating to specific primitives:

```
s.fixedwait -- colon must be present after time value
 current:
  s.fixedwait (100)
 should be:
  s.fixedwait (100:)

s.forcons -- value for max loop count must be moved
             from criteria section to argument section
 current:
  s.forcons   (COUNT,@C02,@C04,@03,@04:8,8,8,120)
 should be:
  s.forcons   (COUNT,@C02,@C04,@03,@04,120:8,8,8)

s.whilestart and s.whilecon -- value for max loop count
              must be moved from s.whilecon to s.whilestart
              and correct precision placed in s.whilecon
 current:
  s.whilestart(@03:)
  s.whilecon  (@T01,@04:4)
 should be:
  s.whilestart(@03,4:)
  s.whilecon  (@T01,@04:8)

s.stboolwait and s.boolwait -- time period should be moved
               from s.boolwait to s.stboolwait
 current:
  s.stboolwait(@03:)
  s.boolwait  (@T01,@03,@04:8,1700)
 should be:
  s.stboolwait(@03,1700:)
  s.boolwait  (@T01,@03,@04:8)

s.waitleast -- many new arguments should be added and CSDL
               requires modification; time period must be
               moved from criteria section to argument
```

section and precision for integer variable must be moved from argument section to criteria section

current:

```
 s.waitleast (@T01,8:500)
```

should be:

```
 s.waitleast (@T02,@T01,@05,@06,500,1500:8,8)

              1    2   3   4   5    6
```

1 -- variable to be used for loop counter
2 -- variable containing integer result
3 -- top label
4 -- bottom label
5 -- time period
6 -- max allowed time period

2. Translator code should be modified to handle time units less than milliseconds since CSDL allows time units as small as nanoseconds.

3. Consideration might be given to modify the Translator to handle fractional numbers vice just integers. Primitives in the Realization Volume would also require modification if this were implemented.

4. The Translator's decision point for specifying 16 bit constants vice 8 bit constants should be between 127 and 128, not the current 128 and 129.

5. The Translator should be modified to always specify an 8 bit precision for the boolean variable in a relational primitive, such as s.eq, regardless of the precisions specified for the other arguments.

6. Variables specified as type DUPLEX must be usable in the CSDL statements, SENSE or ISSUE. Currently the Translator generates syntax errors if this is attempted.

7. The primitive, s.not, can not be produced by the Translator even though it is a valid primitive according to Carson, author of the Tranalator.

8. If the CSDL statement for a <SIMPLE DO> is placed in the contingency list section of a CSDL problem, the Translator generates syntax errors. An example is 'DO MANUAL 4', which means the task MANUAL, is fourth in priority relative to other tasks listed in the contingency list section of a CSDL problem.

9. Syntax error messages must be made easier to understand. An example is the message "expected symbol list follows", where the expected symbol list that follows is a meaningless string of letters and other characters.

10. CSDL and the Translator should be modified to allow more than one expression between s.stboolwait and s.boolwait when a boolean wait construct is specified by a designer. This would make it possible for the condition being checked during a boolean wait, to change.

## LIST OF REFERENCES


1. Booch, G., Software Engineering With ADA, Benjamin/ Cummings Pub. Co., 1983.

2. Boehm, B. W., "Software Engineering: R & D Trends and Defense Needs," Research Directions in Software Technology, 1977.

3. Altman, L. and Scrupski, S. E., editors, Applying Microprocessors, McGraw-Hill, 1976.

4. Ross, A. A., Computer Aided Design of Microprocessor-Based Controllers, Ph.D. Thesis, University of California, Davis, 1978.

5. Lawerence Livermore Laboratory Report pre-print UCRL-78651, Automating the Design of Dedicated Real Time Control Sytems, by M. N. Matelan, 21 Aug 1976.

6. Smith, Jr., T. J., Implementation of a Zilog Z-80 Base Realization Library for the Computer Systems Design Environment, M.S. Thesis, U.S. Naval Postgraduate School, Monterey, California, March 1984.

7. Cetel, A. J., Implementation of an Intel 8086-Based Realization Library for the Control System Design Environment, M.S. Thesis, U.S. Naval Postgraduate School, Monterey, California, June 1984.

8. Carson, T. H., An Input Translator for a Computer-Aided Design System, M.S. Thesis, U.S. Naval Postgraduate School, Monterey, California, June 1984.

9. Riley, R. P., Control System Design Language Implementation of a Gas Turbine Starting Controller, M.S. Thesis, U.S. Naval Postgraduate School, Monterey, California, June 1984.

10. Lawrence Livermore Laboratory Report UCID-17318, Methodology and Planning for a Microprocessor-Oriented Real Time Controller Design Automation System, by M. N. Matelan and R. J. Smith,II, 4 November 1976.

11. Zaks, R., From Chips to Systems: An Introduction to Microprocessors, Sybex, 1981.

12. Poole, J., The CSDE Network, M.S. Thesis, U.S. Naval Postgraduate School, Monterey, California, March 1985.

13. Pro-Log Corporation, STD Bus Technical Manual and Product Catalog, August 1982.

14. Zaks, R., Programming the Z80, Sybex, 1982.

15. Pro-Log Corporation, 7303 Keyboard/Display Card User's Manual, 1981.

16. Lipschutz, S., Essential Computer Mathematics, McGraw-Hill, 1982.

INITIAL DISTRIBUTION LIST

| | | No. Copies |
|---|---|---|
| 1. | Defense Technical Information Center<br>Cameron Station<br>Alexandria, Virginia 22314 | 2 |
| 2. | Library, Code 0142<br>Naval Postgraduate School<br>Monterey, California 93943 | 2 |
| 3. | LTC Alan Ross, Code 52RS<br>Naval Postrgraduate School<br>Monterey, California 93943 | 4 |
| 4. | Prof. Herschel H. Loomis, Code 62LM<br>Naval Postgraduate School<br>Monterey, California 93943 | 1 |
| 5. | LT Robert R. Vogel<br>C/O Fowler<br>2483 Trentwood Blvd.<br>Orlando, Florida 32812 | 1 |
| 6. | Computer Technology Programs Office<br>Code 37<br>Naval Postgraduate School<br>Monterey, California 93943 | 1 |

213184

Thesis
V782
c.1

Vogel
Software maintenance relating to the Translator and Z80 Realization Volume of the Computer Systems Design Environment.

4 SEP 90 36978

213184

Thesis
V782
c.1

Vogel
Software maintenance relating to the Translator and Z80 Realization Volume of the Computer Systems Design Environment.

AMERICAN BINDERIES 2

thesV782
Software maintenance relating to the inp
3 2768 000 61605 6
DUDLEY KNOX LIBRARY